UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

PROYECTO FIN DE CARRERA

Red de alta disponibilidad para centralitas Asterisk basada en DNS SRV y DUNDi. Implementación de un sistema de supervisión

Autor:
Elías Ferreras Sarmiento

Red de alta disponibilidad para centralitas Asterisk basada en DNS SRV y DUNDi. Implementación de un sistema de supervisión

AUTOR: Elías Ferreras Sarmiento
TUTOR: Mario Muñoz Organero
DIRECTOR: José Manuel Ruiz de Marcos

Ingeniería Técnica de Telecomunicación
Especialidad Telemática
Escuela Politécnica Superior
Universidad Carlos III de Madrid

PROYECTO FIN DE CARRERA

Título: *Red de alta disponibilidad para centralitas Asterisk basadas en DNS SRV y DUNDi con sistema de supervisión*

Autor: D. Elías Ferreras Sarmiento

Tutor: D. Mario Muñoz Organero

Director: D. José Manuel Ruiz de Marcos

Agradecimientos

Nunca he sido hombre de muchas palabras y después de veintitrés años probablemente me cuesta cambiar esta mala faceta, pero creo que la ocasión lo merece y haré un pequeño esfuerzo por agradecer a todas aquellas personas que de un modo u otro han hecho posible que este proyecto fin de carrera sea realizado y por ello dar por concluida mi carrera.

En primer lugar y aunque suene muy a tópico, tengo que darles las gracias a mis padres, ya que si no es por un carnet de conducir, hoy no estaría yo aquí terminando mi carrera (ni si quiera la habría empezado). A mi hermana, por tener la suficiente paciencia cuando la he dejado sin Internet para poder probar mis "dichosas centralitas". A mis compañeros de carrera, especialmente a Silvia, compañera y amiga, por tantas horas que hemos pasado en los sótanos del Torres Quevedo programando todas y cada una de las prácticas de la carrera, desde Representación de datos hasta Tratamiento Digital de la Información (sí, más conocida como TDI). Mi compañero "Chemi", por todos los paseos que te has dado por mí para dejarme los libros necesarios para llevar a cabo este proyecto fin de carrera, entre otras cosas. A mis amigos José Ángel y Nico, porque aún viviendo en un mundo donde creen que los ingenieros recién titulados somos una especie de héroes (aún me pregunto de dónde sacan ese tipo de cosas) siempre motiva ver que se sienten orgullosos de ti. Y por último y no menos importante, si no todo lo contrario, a mi gran apoyo estos últimos cinco años, la persona que siempre ha estado y espero esté a mi lado, la mejor persona que conozco, aquella que sin tener ni idea de lo que le cuento, pone el mismo empeño en escucharme como si de un viaje planificado a París entre ella y yo se tratase. Gracias Sandra.

Índice de contenidos

Índice de figuras

Índice de Tablas

Capítulo 1

Finalidad del documento

A lo largo de este documento se pretende dar al lector una visión teórica y técnica de todos los elementos involucrados en la realización del proyecto fin de carrera. Con esto nos referimos a las tecnologías implicadas y en especial a la parte física del proyecto, es decir, las aplicaciones utilizadas para el correcto funcionamiento de éste. Por esta razón, se ha dividido el documento en diferentes capítulos y apartados que se pasan a explicar a continuación.

Capítulo 2

En este capítulo se pretende explicar la necesidad que induce a la realización de este proyecto fin de carrera, es decir, cuáles son los motivos que han generado la demanda por parte del sector que lleva a plantearse la necesidad de crear una red de alta disponibilidad para centralitas Asterisk. Así mismo, se darán a conocer los diferentes objetivos que se persiguen al realizar el presente proyecto.

Capítulo 3

El siguiente capítulo es el conocido como Estado del Arte. A lo largo de sus páginas se dará una visión teórica del proyecto, en lo que a tecnologías implicadas se refiere. De este modo, se hablará de la VoIP (voz sobre IP) y todo lo que ésta implica. Además, hay que recordar que el proyecto fin de carrera cuenta con la realización de un sistema de supervisión que permite monitorizar el clúster de centralitas Asterisk, por lo que se explica la tecnología utilizada para tal cometido, en este caso Java. Para terminar con este apartado, se explican las diferentes soluciones existentes en el mercado que tengan relación directa con el proyecto.

Capítulo 4

Este es el capítulo más importante del documento. En él y en sus distintos apartados, se le dará al lector la oportunidad de conocer cómo se ha llevado a cabo el proyecto fin de carrera. Esto se traduce en una explicación detallada de todos los módulos que componen el proyecto. Así, podrá conocer las diferentes configuraciones necesarias para la implementación de la red y el estilo utilizado para la programación del sistema de supervisión.

Capítulo 5

El siguiente capítulo muestra el material utilizado para poner en práctica tanto la red de alta disponibilidad como el sistema de supervisión, así como las diferentes pruebas realizadas para verificar el correcto funcionamiento del proyecto fin de carrera realizado.

Capítulo 6

Este capítulo muestra información acerca del coste que ha llevado a cabo realizar el proyecto fin de carrera y una posible inversión del mismo. Adicionalmente se mostrará información acerca de las tareas y duración de éstas para la ejecución del proyecto.

Capítulo 7

Para terminar, en este último capítulo se muestra una línea de futuro sobre el proyecto fin de carrera, asumiendo éste como un producto dentro del mercado. De esta forma, se darán a conocer diferentes líneas que permitan una evolución del mismo.

Capítulo 2

Objetivo del proyecto

A lo largo de los años en que Asterisk se ha convertido en referente mundial a lo que PBX IP se refiere, muchos han sido los que han solicitado la necesidad de crear un posible clúster para Asterisk. A fecha de hoy, sigue siendo una de las prioridades de la empresa Digium, responsable directo del desarrollo de Asterisk. Por tanto, uno de los primeros objetivos en este sentido es conseguir propagar la información de los usuarios (libres, ocupados, hablando, no disponible, etc.) entre los distintos servidores que forman el clúster. Para ello, el equipo de desarrolladores de Asterisk está utilizando un framework especial para programar en este tipo de infraestructuras llamada OpenAIS y así han creado un nuevo módulo llamado res_ais que permite controlar el estado de una extensión situada en otro Asterisk perteneciente a uno de los nodos del clúster. El siguiente paso será propagar esta información a través de Asterisk conectados entre sí por el protocolo DUNDi.

Como se podrá observar a lo largo del documento, el proyecto fin de carrera trata de construir un clúster de Asterisk haciendo uso no sólo del protocolo DUNDi, sino también de un servidor DNS, utilizando registros de recursos SRV y de bases de datos para almacenar información relevante a cada Asterisk, como pueden ser los usuarios registrados en cada uno de ellos. Sabiendo esto, se podría decir que el objetivo principal del proyecto es conseguir un sistema estable y robusto de centralitas Asterisk que nos permitan alcanzar una red de alta disponibilidad. Explicado de forma más concisa, el objetivo es conseguir un clúster de Asterisk que permitan dar soporte a los usuarios aun cuando una de las centralitas esté fuera de servicio. No obstante, para controlar que todo funciona correctamente, se programará una aplicación web con el lenguaje de programación Java a modo de sistema de supervisión. Sin duda, esto último es la innovación dentro del proyecto, ya que el clúster de Asterisk se desarrollará con funcionalidades propias de este software.

Pero, ¿qué pretendemos cuando nos referimos a implementar una red de alta disponibilidad para Asterisk? A esta pregunta se responde con un sencillo ejemplo de funcionamiento de dicha centralita:

Un usuario registra un terminal contra su centralita Asterisk, por tanto, y si todo es correcto, este usuario puede comenzar a realizar llamadas con su terminal. Pasado un tiempo dicha centralita se cae, y con ello, toda la red telefónica que gestiona dicha centralita, lo que acarrea que, por ejemplo, una pequeña empresa quede incomunicada durante el tiempo que se tarde en restablecer el servicio.

Visto esto, es fácil intuir que nuestra red de alta disponibilidad pretende dar servicio continuo aun cuando una centralita quede inutilizada. Para ello basta con estudiar el tipo de red a tratar y el número aproximado de usuarios o terminales gestionados por la centralita para incorporar un número suficiente de éstas y formar un "clúster de Asterisk". De esta forma se consigue dar respaldo unas a otras.

Pero como ya se comentó, tenemos por finalidad crear un sistema de supervisión mediante una aplicación web que nos permita gestionar dichas centralitas. El objetivo es conseguir administrar desde nuestra aplicación web si las centralitas funcionan de la forma deseada, qué usuarios están registrados en cada momento, las llamadas realizadas...Esto será posible ya que configuraremos Asterisk en RealTime, es decir, en tiempo real, de forma que almacenemos cierta información en una base de datos, en este caso MySQL, y podamos acceder a ella. En apartados posteriores, se dará una descripción mucho más detallada de los módulos que forman parte en esta red de alta disponibilidad y cómo debe llevarse a cabo.

Capítulo 3

Estado del Arte

Como ya se sabe, este proyecto fin de carrera pretende crear un clúster de Asterisk junto con un sistema de supervisión que nos permita monitorizar las centralitas en tiempo real, pero es importante saber qué tecnologías entran en juego para que todo el sistema funcione de forma correcta. Debemos conocer ciertos conceptos como la telefonía IP, voz sobre IP (VoIP) o saber incluso qué es una PBX-IP (también conocido por IP-PBX), sabiendo que se pretende crear una red telefónica sobre VoIP de alta disponibilidad. No obstante, es importante conocer también la tecnología Java, en concreto Servlets y JavaServer Pages, ya que es ésta la que se empleará para el sistema de supervisión.

3.1 Breve historia de la VoIP

Hasta la fecha de hoy, las redes tradicionales que funcionan por conmutación de circuito, como pueden ser las redes RTC (Red Telefónica pública Conmutada) también conocidas como PSTN, o RDSI (Red Digital de Servicios Integrados), han sido utilizadas para el funcionamiento de los servicios de voz, aunque también permiten el envío de datos, gracias a un módem o el actual ADSL. Para las empresas, se podía decir que existían dos redes claramente diferenciadas, una encargada del tráfico y servicios de voz y otra para el tráfico de datos, basándose esta última en una tecnología de conmutación de paquetes. Con las tecnologías actuales de VoIP se tiene la capacidad de proporcionar unos servicios garantizados sobre una única red donde confluyen tráficos tanto de voz como de datos, lo que nos lleva a afirmar que se dispone de una red multiservicio.

La voz sobre redes IP se implementó principalmente para reducir costes y para reducir el ancho de banda requerido por la voz digitalizada a través de las redes convencionales de conmutación de circuitos, que es de 64Kbit/s, mediante el empleo de la conmutación de paquetes asociada a la comprensión vocal, aprovechando los procesos de compresión diseñados para sistemas celulares digitales en la década de los 80. De esta forma, se consiguió reducir el coste en el transporte nacional, pero sobre todo y quizá más importante, debido a su elevado precio, el internacional. Luego se aplicó sobre las redes LAN e Internet, siendo esta última la más importante puesto que las llamadas se consideraban locales y por tanto de un coste reducido.

Hoy en día, los usuarios utilizan para sus comunicaciones vocales tanto la red telefónica pública conmutada o la Red Digital de Servicios Integrados, como la comunicación a través de redes móviles, como GSM o UMTS. Los usuarios acceden al servicio telefónico fijo por medio del bucle de abonado, que comunica la central telefónica pública con su domicilio, bien sea residencial o empresarial. En el caso de las redes móviles, lo hacen a través de las estaciones base de radio y sus teléfonos móviles. El caso de las empresas tiene sus particularidades de forma que actualmente, la mayoría de las empresas tienen su propia red telefónica convencional, diseñada sobre PBX, que soportan todos los servicios telefónicos tradicionales. Hay que recordar que estos

sistemas PBX se conectan a la RTC o RDSI para llamadas externas a teléfonos fijos o móviles y que, si la empresa tiene un gran volumen de llamadas y varias oficinas, éstas pueden conectarse por líneas dedicadas de alta capacidad, que le reducirán el coste considerablemente. Otra de las características de los sistemas de comunicación en la empresa es tener su propia red de datos de área local (LAN), donde se conectan sus diversos equipos de datos. Estas redes de datos se han ido ampliando, pudiendo conectar equipos en oficinas remotas mediante la interconexión de diversas LAN, por medio de líneas dedicadas. Este tipo de redes se conocen como VPN (Virtual Private Network). El hecho de tener dos redes independientes para comunicaciones telefónicas y de datos es algo caro e innecesario, por lo que la telefonía IP proporciona una nueva vía en el campo de las comunicaciones de la empresa.

Actualmente se habla mucho de dos conceptos, telefonía IP y VoIP, y se tiende a utilizar ambos términos para lo mismo, cuando existe una diferencia entre ellos.

La telefonía IP se refiere a la utilización de una red IP (privada o pública, como es Internet) por la que transmitimos los servicios de voz, fax y mensajería. Esta red IP puede ser utilizada para realizar las llamadas internas de la propia empresa, así como las llamadas externas, usando, por ejemplo, Internet en lugar de la red telefónica pública conmutada.

La VoIP es la tecnología usada para el funcionamiento de la telefonía IP. VoIP gestiona el envío de información de voz utilizando IP (Internet Protocol). La información analógica vocal se transforma en paquetes digitales diferenciados que se envían por la red. Los paquetes de información de voz viajan por la red IP, del mismo modo que los datos generados por una comunicación de correo electrónico, por ejemplo.

Por tanto, la telefonía IP es una aplicación inmediata de esta tecnología, de manera que permite la realización de llamadas telefónicas ordinarias sobre las redes IP u otras redes de paquetes, utilizando ordenadores personales (PC), gateways, gatekeepers, unidades de multiconferencia o teléfonos normales. En general, soporta servicios de comunicación de voz, fax, de mensajes de voz, etc. que se transportan vía redes IP, como Internet, en lugar de ser transportados vía la red telefónica convencional.

3.2 Telefonía IP frente a la telefonía convencional

Pero, ¿qué diferencia existe entre la telefonía IP y la convencional? Pues bien, en una llamada telefónica "normal" la central establece una conexión permanente entre ambos interlocutores, conexión que se utiliza para llevar las señales de voz, esto es una comunicación conmutada por circuitos. En una llamada telefónica por IP, los paquetes de datos, que contienen la señal de voz digitalizada y comprimida, se envían a través de la red IP a la dirección IP del destinatario. Cada paquete puede utilizar un camino para llegar al destino, puesto que están compartiendo un medio, en este caso una red de datos. Cuando llegan a su destino son ordenados y convertidos de nuevo a señales de voz. Esta comunicación se conoce como conmutación de paquetes. Por otro lado, una llamada telefónica convencional, requiere una enorme red de centralitas telefónicas conectadas entre sí mediante fibra óptica, satélites de telecomunicaciones, además de la conexión directa del bucle de abonado (teléfono del usuario) con su central telefónica local. Esto supone inversiones para crear y mantener esa infraestructura, que se traducen

en mayor coste de llamada al usuario especialmente cuando las llamadas son de larga distancia. Por el contrario, en una llamada telefónica IP estamos comprimiendo la señal de voz y utilizamos una red de paquetes sólo cuando es necesario. Los paquetes de datos de diferentes llamadas, e incluso de diferentes tipos de datos, pueden viajar por la misma línea al mismo tiempo. Además, el acceso a Internet cada vez es más barato, muchos ISP lo ofrecen gratis, sólo se paga la llamada, siempre con tarifa local o nacional extendiéndose además las tarifas planas, conexiones por cable, etc. con lo que el ahorro de costes es aún mayor.

Es importante aclarar dos términos que han salido en la anterior explicación, en referencia a la comunicación por conmutación de circuitos y la conmutación de paquetes.

3.2.1 Conmutación de circuitos

La conmutación de circuitos como ya sabemos se usa en redes telefónicas públicas. La técnica de conmutación de circuitos se desarrolló para tráfico de voz aunque también puede gestionar tráfico de datos de forma no muy eficiente. En la conmutación de circuitos se establece un canal de comunicaciones dedicado entre dos estaciones donde se reservan recursos de transmisión y de conmutación de la red para su uso exclusivo en el circuito durante la conexión. La transmisión es transparente, ya que, una vez establecida la conexión parece como si los dispositivos estuviesen directamente conectados.

Diversos aspectos importantes de las redes de conmutación de circuitos han cambiado de forma drástica con el incremento de la complejidad y digitalización de las redes de telecomunicaciones públicas, haciendo que las técnicas de encaminamiento jerárquico hayan sido reemplazadas por otras no jerárquicas, más flexibles y potentes, que permiten mayor eficiencia y flexibilidad.

Las comunicaciones mediante conmutación de circuitos implican la existencia de un camino o canal de comunicación dedicado entre dos estaciones, que es una secuencia de enlaces conectados entre nodos de la red. En cada uno de los enlaces físicos se dedica un canal lógico para cada conexión establecida. Por tanto, se puede decir que la conmutación de circuitos implica tres fases:

- **Establecimiento del circuito**
- **Transferencia de datos**
- **Desconexión del circuito**

Pero debido al auge de las transmisiones de datos, la conmutación de circuitos es un sistema muy ineficiente ya que mantiene las líneas ocupadas por mucho tiempo aun cuando no hay información circulando por ellas. Además, la conmutación de circuitos requiere que los dos sistemas conectados trabajen a la misma velocidad, cosa que no suele ocurrir hoy en día debido a la gran variedad de sistemas que se comunican.

3.2.2 Conmutación de paquetes

En conmutación de paquetes, los datos se transmiten en paquetes cortos. Para transmitir grupos de datos más grandes, el emisor trocea estos grupos en paquetes más pequeños y les adiciona una serie de bits de control. En cada nodo, el paquete se recibe, se almacena durante un cierto tiempo y se transmite hacia el emisor o hacia un nodo intermedio.

Las ventajas de la conmutación de paquetes frente a la de circuitos son:

- La eficiencia de la línea es mayor ya que cada enlace se comparte entre varios paquetes que estarán en cola para ser enviados en cuanto sea posible. En conmutación de circuitos, la línea se utiliza exclusivamente para una conexión, aunque no haya datos a enviar.

- Se permiten conexiones entre estaciones de velocidades diferentes. Esto es posible ya que los paquetes se irán guardando en cada nodo conforme lleguen (en una cola) y se irán enviando a su destino.

- No se bloquean llamadas ya que todas las conexiones se aceptan, aunque si hay muchas, se producen retardos en la transmisión.

- Se pueden usar prioridades porque un nodo puede seleccionar de su cola de paquetes en espera de ser transmitidos, aquellos más prioritarios según ciertos criterios de prioridad.

Cabe mencionar también que existen diferentes técnicas para llevar a cabo la conmutación de paquetes. Cuando un emisor necesita enviar un grupo de datos mayor que el tamaño fijado para un paquete, éste los trocea en paquetes más pequeños (segmentación de paquetes, característica del protocolo IP) y los envía uno a uno al receptor. Hay dos técnicas básicas para el envío de estos paquetes:

- **Técnica de datagramas:** cada paquete se trata de forma independiente, es decir, el emisor enumera cada paquete, le añade información de control (por ejemplo número de paquete, nombre, dirección de destino, etc...) y lo envía hacia su destino. Puede ocurrir que por haber tomado caminos diferentes, un paquete con número por ejemplo 6 llegue a su destino antes que el número 5. También puede ocurrir que se pierda el paquete número 4. Todo esto no lo sabe ni puede controlar el emisor, por lo que tiene que ser el receptor el encargado de ordenar los paquetes y saber los que se han perdido (para su posible reclamación al emisor), y para esto, debe tener el software necesario.

- **Técnica de circuitos virtuales:** antes de enviar los paquetes de datos, el emisor envía un paquete de control que es de Petición de Llamada, este paquete se encarga de establecer un camino lógico de nodo en nodo por donde irán uno a uno todos los paquetes de datos. De esta forma se establece un camino virtual para todo el grupo de paquetes. Este camino virtual será numerado o nombrado inicialmente en el emisor y será el paquete inicial de Petición de Llamada el encargado de ir informando a cada uno de los nodos por los que pase de que más adelante irán llegando los paquetes de datos con ese nombre o número. De esta

forma, el encaminamiento sólo se hace una vez (para la Petición de Llamada). El sistema es similar a la conmutación de circuitos, pero se permite a cada nodo mantener multitud de circuitos virtuales a la vez.

Las ventajas de los circuitos virtuales frente a los datagramas son:

- El encaminamiento en cada nodo sólo se hace una vez para todo el grupo de paquetes, por lo que los paquetes llegan antes a su destino.

- Todos los paquetes llegan en el mismo orden del de partida ya que siguen el mismo camino.

- En cada nodo se realiza detección de errores, por lo que si un paquete llega erróneo a un nodo, éste lo solicita otra vez al nodo anterior antes de seguir transmitiendo los siguientes.

Las desventajas de los circuitos virtuales frente a los datagramas**:**

- En datagramas no hay que establecer llamada (para pocos paquetes, es más rápida la técnica de datagramas).

- Los datagramas son más flexibles, es decir que si hay congestión en la red una vez que ya ha partido algún paquete, los siguientes pueden tomar caminos diferentes (en circuitos virtuales, esto no es posible).

- El envío mediante datagramas es más seguro ya que si un nodo falla, sólo unos paquetes se perderán (en circuitos virtuales se perderán todos).

Con todo esto, podemos decir que la característica principal en redes que emplean la conmutación de circuitos es la utilización por cada usuario de un ancho de banda para su comunicación, mientras que en las que usan conmutación de paquetes, el ancho de banda es compartido por todos los usuarios que utilicen la red en un momento determinado.

Una vez se conocen las diferencias entre la telefonía convencional y la telefonía IP, se puede pasar a explicar cómo funciona la VoIP, que es el tema que verdaderamente nos concierne.

3.3 Funcionamiento de VoIP

Realmente, se podría decir que la VoIP es simplemente la transferencia de las conversaciones de voz convertidas en datos sobre una red IP (pública o privada) que si dispone de un gran ancho de banda puede dar una buena calidad. Como ya se ha explicado, esta tecnología utiliza para su comunicación la conmutación de paquetes. Cada paquete contiene la información de direccionamiento en la que se especifica la dirección del equipo origen y destino (parte de la cabecera del protocolo IP). Los paquetes dentro de una simple transmisión pueden tomar diversas vías desde el punto de origen al punto final de destino a través de una red de datos.

En la secuencia de la fase de establecimiento en una llamada de VoIP, se han de simular los tonos de: invitación a marcar, de llamada y ocupado. La misma información de audio de la llamada necesita ser transformada de analógico a digital en el origen, ser fraccionada en paquetes y ser enviada a través de la red en el formato de los paquetes. A la llegada de estos paquetes al destino se ha de proceder de forma inversa, para ser convertidos de nuevo de digital a analógico. La función de los codificadores (codecs) en ambos extremos es la conversión analógica a digital y viceversa.

Como se puede intuir, en la telefonía IP el cambio fundamental se produce en la red de transporte, ya que ahora esta tarea se lleva a cabo por una red basada en el protocolo IP, como por ejemplo Internet. En cuanto a la red de acceso, puede ser la misma que en la telefonía convencional, físicamente hablando.

Los elementos necesarios para que se puedan realizar llamadas vocales a través de una red IP depende en gran medida de qué terminal se utiliza en ambos extremos de la conversación. Éstos pueden ser terminales IP, como pueden ser los softphones, o no IP, que serían los teléfonos analógicos convencionales. Hay que señalar que en el caso de que uno de los terminales sea no IP, la comunicación pasará a través de una red de datos como por la RTC, Red Telefónica pública Conmutada. Pero debemos conocer cuáles son los elementos que componen la VoIP, ya que es de intuir que serán distintos a los de la telefonía convencional. A continuación nombramos todos ellos, pero sólo se explicarán en las siguientes líneas los más importantes en relación al proyecto fin de carrera. Éstos son:

- Codificadores (codecs)
- Protocolos
- Servidores y PBX de telefonía IP
- Gateways y Routers IP
- Teléfonos y softphones IP

Como se puede apreciar, se dispone de cinco elementos para poner en funcionamiento la VoIP. De todos ellos, a continuación sólo se explicarán los tres últimos, para obtener más información acerca de los protocolos y codecs utilizados se puede consultar el ANEXO A y B, respectivamente.

3.3.1 Servidores y PBX de telefonía IP

Muchos de los intercambios de datos en una red están basados en el concepto de servidor-cliente. El ordenador cliente realiza una petición de servicio al ordenador servidor, en el que se resuelven éstos y se devuelven los resultados.

El añadir voz a las redes IP proporciona la utilización de otro tipo de servidores diseñados para la realización de los servicios de la voz de innovadoras maneras. Una PBX-IP sirve generalmente como el servidor usado en la telefonía IP. Pero lo primero de todo, ¿qué es una PBX?

Una PBX o Private Branch Exchange, es un equipo que tiene control por software y proporciona funciones de conmutación a los usuarios a ella conectados. La PBX les permite conmutar sus llamadas internas sin necesidad de acceder a la red pública de conmutación y la operadora es la encargada de atender las llamadas entrantes y dar curso a las salientes. Comúnmente puede tener desde dos extensiones a diez mil, junto con una conexión directa con la red tradicional de telefonía, ya sea RTC o RDSI, para llamadas hacia y desde el exterior.

Las PBX son en gran medida similares a las centralitas públicas, excepto en que normalmente no incluyen algunas de las funciones operacionales y administrativas, como por ejemplo las protecciones de línea o la redundancia de sus elementos. Ambas constan de dos partes claramente definidas: la unidad de conmutación y la unidad de control, siendo en términos generales, la primera la encargada de establecer el canal físico para poner a los usuarios en comunicación, y la segunda, la de atender la señalización entrante y saliente, procesar las señales recibidas e indicar a la primera qué circuitos interconectar. Se podría profundizar mucho más en el tema de las PBX, pero para comprender la finalidad del proyecto esto nos da una idea general de qué es una PBX.

Por tanto, una vez que sabemos qué es una PBX, es fácil intuir qué es una PBX-IP. Ésta última es una centralita que soporta el uso de VoIP que puede construirse sobre una plataforma de PC que funciona con sistema operativo que puede ser Microsoft Windows, Linux o Sun Solaris. Mientras que las PBX tradicionales de RTC/RDSI ofrecen múltiples servicios desarrollados durante décadas, como el desvío de llamada, la transferencia, etc. las PBX-IP están proporcionando rápidamente los mismos e incluso superiores servicios.

Con la telefonía IP aparecen nuevos conceptos, que junto a los servidores se agrupan y juntos ofrecen ampliaciones seguras. Los servidores agrupados se pueden manejar como un solo servidor. Entre los nuevos conceptos aparece el de Gatekeeper, que es otro tipo de servidor. Éste es un dispositivo que proporciona funciones de autenticación, registro y conversión de las direcciones IP a direcciones telefónicas. El protocolo que utilizan los Gatekeeper es el H.323 para proporcionar características de control en la admisión y otras funciones para el manejo de los servidores multimedia. Principalmente, este servidor realiza el control para el procesamiento de la llamada en el protocolo H.323. Es un software que puede funcionar sobre las plataformas anteriormente citadas. Pueden existir varios Gatekeeper por razones de redundancia y compartir la carga de la red. El principal parámetro de este servidor es la cantidad de llamadas cursadas en las horas pico. Dicho parámetro se conoce como BHCA (Busy Hour Call Attempts). Las funciones del Gatekeeper son:

- Traslación de direcciones desde una dirección alias del terminal hacia una dirección de capa ¾ (socket).

- Control de admisión para autorizar el acceso a la red mediante mensajes ARQ/ACF/ARJ (protocolo RAS).

- Control de ancho de banda mediante mensajes BRQ/BRJ/BCF (protocolo RAS).

- Señalización de control de llamada para autorización o rechazo de llamadas.

- Servicios de directorio.
- Servicios de asignación/reserva de ancho de bando

3.3.2 Gateways y routers

Estos dos conceptos también aparecen en VoIP. Los gateways proporcionan conectividad entre el mundo IP y el de telefonía convencional. Realizan la emulación de interfaz FXO/FXS (Foreign Exchange Office/Station), lo que permite adaptar una PBX a la VoIP. Se conecta a la PBX convencional por un lado y a la red de transporte IP por el otro, lo que permite conectar un usuario convencional a la red de Telefonía IP pública. Permite la traslación de direcciones desde IP a la UIT E.164 de la red telefónica convencional. Es decir, actúa de interfaz desde la red IP (dirección de 4 bytes, si es IPv4) hacia la RTC (dirección de 16 dígitos decimales).

Los GW-E1 se encuentran entre la red IP y la RTC para interconectar distintos proveedores de telefonía mediante técnicas de transporte diversas. Entre las funciones de GW (Gateway) más importantes se encuentran.

- La conversión de codificación vocal.
- La supresión de silencios y señalización DTMF.
- La supresión de ecos.
- Generar las conexiones RTP.

Los routers son dispositivos de conexión en la red IP que encaminan los datagramas basándose en las direcciones de red que llevan éstos en las cabeceras, operando con protocolos IP.

Por el examen de las cabeceras del paquete IP, se mueven los datagramas de voz RTP en gateways y routers a través de una red IP. Los routers toman las decisiones necesarias para mover los paquetes de un router a otro, a lo largo de la trayectoria hasta llegar al destino. Los gateways proporcionan en VoIP la conexión entre la red de VoIP y la tradicional RTC/RDSI. Estos dispositivos desempeñan un papel predominante en la trayectoria de la migración hacia VoIP. Al existir poco equipamiento VoIP puro hoy en día, es necesario conectar con las redes RTC/RDSI para poder establecer las conexiones con los usuarios de esas redes. Los gateways también proporcionan la conversión entre distintos codecs. Si en redes RTC/RDSI se utiliza un códec G.711 y en la red de VoIP se puede utilizar un códec G.729, las informaciones de voz deben ser convertidas a G.711 antes de ser transferidas a la red telefónica convencional.

Si recordamos unas líneas más arriba, se comentó que se hablaría de servidores, PBX-IP y gateways y routers por tener relación directa con el proyecto fin de carrera. Pues bien, esto se debe a que una vez se conocen estos conceptos, se puede entender ahora el por qué de utilizar para las centralitas del clúster el software Asterisk.

3.4 Asterisk

Asterisk es una centralita software (PBX) de código abierto. Como cualquier centralita PBX permite interconectar teléfonos y conectar dichos teléfonos a la red telefónica convencional. Su nombre viene del símbolo asterisco (*) en inglés.

El creador original de esta centralita es Mark Spencer de la compañía Digium que sigue siendo el principal desarrollador de las versiones estables. Pero al ser de código libre, existen multitud de desarrolladores que han aportado funciones y nuevas aplicaciones. Originalmente fue creada para sistemas Linux pero hoy en día funciona también en sistemas OpenBSD, FreeBSD, Mac OS X, Solaris Sun y Windows. Pero Linux sigue siendo la que más soporte presenta.

El paquete básico de Asterisk incluye muchas características que antes sólo estaban disponibles en caros sistemas propietarios como creación de extensiones, envío de mensajes de voz a e-mail, llamadas en conferencia, menús de voz interactivos o distribución automática de llamadas. Además se pueden crear nuevas funcionalidades mediante el propio lenguaje de Asterisk o módulos escritos en C o mediante scripts AGI escritos en Perl o en otros lenguajes. A continuación se muestran los servicios más importantes que ofrece asterisk.

- Operadora automática/virtual: Permite pasar la llamada entrante a la extensión deseada mediante menús interactivos sin la necesidad de un operador físico. Esto es posible gracias a un sistema basado en reconocimiento de voz y/o tonos DTMF (Dual Tone Multi Frequency) generados al marcar en el teclado del teléfono. Este servicio automatizado permite atender a múltiples llamadas simultáneas.

- Buzón de Voz: Es un servicio de almacenamiento de mensajes de voz.

- Marcación rápida a números de emergencia como el 112, policía o bomberos.

- Desvío de llamada a otro Terminal en el caso de que la extensión se encuentre ocupada o no responda.

- Transferencia de llamadas a distintas extensiones.

- Follow-me: Listado de números para redireccionar en caso de que la extensión esté ocupada.

- Llamada en espera: Mantener una conversación en espera para atender nuevas llamadas.

- Música en espera: Servicio de reproducción de música antes de atender llamadas entrantes.

- Tarificación de llamadas: Sistema para el cálculo de los costes de las llamadas.

- CallerID: Servicio para la identificación del número de llamada entrante.

- Salas de conferencia: Servicio que permite mantener una conversación entre más de dos terminales.
- Listas negras: Restricción del acceso a ciertos números.
- Registro y listado de llamadas.
- Envío o recepción automática de faxes.
- Grabación de llamadas.
- Integración con bases de datos.
- Mensajería SMS.
- Monitorización de llamadas en curso.
- Distribuidor automático de tráfico de llamadas.

Para poder utilizar teléfonos convencionales en un servidor Linux corriendo Asterisk o para conectar a una línea de teléfono analógica se suele necesitar hardware especial (no vale con un modem ordinario). Digium y otras compañías venden estas tarjetas.

Pero quizás lo más interesante es que Asterisk soporta numerosos protocolos de VoIP como SIP y H.323. Asterisk puede operar con muchos teléfonos SIP, actuando como "registrar" o como "gateway" o entre teléfonos IP y la red telefónica convencional. Los desarrolladores de Asterisk han diseñado un nuevo protocolo llamado IAX para una correcta optimización de las conexiones entre centralitas Asterisk.

Al soportar una mezcla de la telefonía tradicional y los servicios de VoIP, Asterisk permite a los desarrolladores construir nuevos sistemas telefónicos de forma eficiente o migrar de forma gradual los sistemas existentes a las nuevas tecnologías. Algunos sitios usan Asterisk para reemplazar a antiguas centralitas propietarias, otros para proveer funcionalidades adicionales y algunas otras para reducir costes en llamadas a larga distancia utilizando Internet.

Después de tener una visión más concreta y técnica de las tecnologías implicadas en el clúster de Asterisk, se puede continuar con la otra parte del proyecto. Como ya se ha comentado, se pretende crear un sistema de supervisión mediante una aplicación web que permita monitorizar las centralitas en tiempo real. Para ello, el lenguaje de programación utilizado será Java y dentro de dicho lenguaje, se utilizará una estructura cliente-servidor basada en Servlets y JavaServer Page (JSP). De este modo, se explicará en qué consiste esta tecnología.

3.5 Servlets y JavaServer Page (JSP)

Los Servlets son la respuesta de la tecnología Java a la programación CGI. Como aclaración, decir que la programación CGI, que viene de Common Gateway Interface, fue de las primeras formas de programación web dinámica. Los Servlets son programas que se ejecutan en un servidor Web y construyen páginas Web. Construir páginas Web al vuelo (o dinámicamente) es útil por un número de razones:

- La página Web está basada en datos enviados por el usuario. Por ejemplo, las páginas de resultados de los motores de búsqueda se generan de esta forma, y los programas que procesan pedidos desde sites de comercio electrónico también.

- Los datos cambian frecuentemente. Por ejemplo, un informe sobre el tiempo o páginas de cabeceras de noticias podrían construir la página dinámicamente, quizás devolviendo una página previamente construida y luego actualizándola.

- Las páginas Web que usan información desde bases de datos corporativas u otras fuentes. Por ejemplo, usaríamos esto para hacer una página Web en una tienda on-line que liste los precios actuales y el número de artículos en stock.

Pero existen diversas razones que llevan a utilizar esta tecnología frente al CGI tradicional. Los Servlets Java son más eficientes, fáciles de usar, más poderosos, más portables, y más baratos que el CGI tradicional y otras muchas tecnologías del tipo CGI. Podemos destacar las siguientes virtudes:

- **Eficiencia**. Con CGI tradicional, se arranca un nuevo proceso para cada solicitud HTTP. Si el programa CGI hace una operación relativamente rápida, la sobrecarga del proceso de arrancada puede dominar el tiempo de ejecución. Con los Servlets, la máquina Virtual Java permanece arrancada, y cada petición es manejada por un thread Java de peso ligero, no un pesado proceso del sistema operativo. De forma similar, en CGI tradicional, si hay N peticiones simultáneas para el mismo programa CGI, el código de este problema se cargará N veces en memoria. Sin embargo, con los Servlets, hay N threads pero sólo una copia de la clase Servlet. Los Servelt también tienen más alternativas que los programas normales CGI para optimizaciones como los cachés de cálculos previos, mantener abiertas las conexiones de bases de datos, etc.

- **Conveniencia**. Los Servlets tienen una gran infraestructura para análisis automático y decodificación de datos de formularios HTML, leer y seleccionar cabeceras HTTP, manejar cookies, seguimiento de sesiones, y muchas otras utilidades.

- **Potencia**. Los Servlets Java nos permiten fácilmente hacer muchas cosas que son difíciles o imposibles con CGI normal. Por algo, los Servlets pueden hablar directamente con el servidor Web. Esto simplifica las operaciones que se necesitan para buscar imágenes y otros datos almacenados en situaciones estándares. Los Servlets también pueden compartir los datos entre ellos, haciendo las cosas útiles como almacenes de conexiones a bases de datos fáciles de implementar. También pueden mantener información de solicitud en

solicitud, simplificando cosas como seguimiento de sesión y el caché de cálculos anteriores.

- **Portable**. Los Servlets están escritos en Java y siguen un API bien estandarizado. Consecuentemente, los Servlets escritos, digamos en el servidor I-Planet Enterprise, se pueden ejecutar sin modificarse en Apache, Microsoft IIS, o WebStar. Los Servlets están soportados directamente o mediante plug-in en la mayoría de los servidores Web.

- **Barato**. Hay un número de servidores Web gratuitos o muy baratos que son buenos para el uso "personal" o el uso en sites Web de bajo nivel. Sin embargo, con la excepción de Apache, que es gratuito, la mayoría de los servidores Web comerciales son relativamente caros. Una vez que tengamos un servidor Web, no importa el coste del servidor, añadirle soporte para Servlets (si no viene pre configurado para soportarlos) es gratuito o muy barato.

Ahora que se sabe qué son los Servlets, continuemos con los JavaServer Pages. Ésta es una tecnología que nos permite mezclar HTML estático con HTML generado dinámicamente. Muchas páginas Web que están construidas con programas CGI son casi estáticas, con la parte dinámica limitada a muy pocas localizaciones. Pero muchas variaciones CGI, incluyendo los Servlets, hacen que generemos la página completa mediante nuestro programa, incluso aunque la mayoría de ella sea siempre lo mismo. JSP nos permite crear dos partes de forma separada.

Pero qué ventajas nos ofrecen los JSP. A continuación se muestran la diferencia frente a distintas tecnologías:

- **Contra Active Server Pages (ASP)**. ASP es una tecnología similar de Microsoft. Las ventajas de JSP están duplicadas. Primero, la parte dinámica está escrita en Java, no en Visual Basic, otro lenguaje específico de MS, por eso es mucho más poderosa y fácil de usar. Segundo, es portable a otros sistemas operativos y servidores Web.

- **Contra los Servlets**. JSP no nos da nada que no pudiéramos en principio hacer con un Servlet. Pero es mucho más conveniente escribir HTML normal que tener que hacer un billón de sentencias println que generen HTML.

- **Contra Server-Side Includes (SSI)**. SSI es una tecnología ampliamente soportada que incluye piezas definidas externamente dentro de una página Web estática. JSP es mejor porque nos permite usar servlets en vez de un programa separado para generar las partes dinámicas. Además, SSI, realmente está diseñado para inclusiones sencillas, no para programas "reales" que usen formularios de datos, hagan conexiones a bases de datos, etc.

- **Contra JavaScript**. JavaScript puede general HTML dinámicamente en el cliente. Esta es una capacidad útil, pero sólo maneja situaciones donde la información dinámica está basada en el entorno del cliente. Con la excepción de las cookies, el HTTP y el envió de formularios no están disponibles con JavaScript. Y, como se ejecuta en el cliente, JavaScript no puede acceder a los

recursos en el lado del servidor, como bases de datos, catálogos, información de precios, etc.

Después de leer esto, puede dar la impresión de que Servlets y JSP pueden hacer lo mismo y sólo existe una diferencia entre ellos, en cuanto a eficiencia de código se refiere. Pues bien, la verdad es que si se utiliza tanto Servlets como JSP en una misma aplicación web, el resultado será una plataforma mucho más eficiente y robusta. Lo único que hay que tener claro es para qué se utilizan los Servlets y para qué las páginas JSP. Dentro de una aplicación web, debemos distinguir dos estados bien distintos; el primero de ellos será el procesado de datos; el segundo la presentación de la información. Para el primero de ellos se utilizarán los Servlets. Un ejemplo sería la conexión con una base de datos. De esta forma, la aplicación envía el control al Servlet que se encargará de conectar y obtener los datos de la base de datos y éste, pasará el control a la página JSP quien se encargará del segundo cometido, es decir, de presentar la información al usuario. Como es lógico, la aplicación web del proyecto fin de carrera está programada de este modo, aunque en el módulo del servidor web del Capítulo 4 de este mismo documento se darán más detalles al respecto.

3.6. Soluciones en el mercado

Ya se conocen todas las tecnologías implicadas en la realización del proyecto fin de carrera, por lo que podríamos preguntarnos qué solución se puede ofrecer al mercado con los productos ya existentes. Lo cierto es que Asterisk, al tratarse de un software libre, puede ser modificado al antojo de cualquier desarrollador de dicha tecnología. Por lo general, los avances de Asterisk se desarrollan en Digium, pero existen muchas empresas que se dedican en exclusiva a la consultoría de Asterisk, por lo que los productos que vendan a sus clientes tengan cierto carácter privativo y se desconozca dentro de la comunidad Asterisk. Como se comentó en las primeras líneas del capítulo 2, el clúster de Asterisk se ha realizado con funciones propias de dicho software, por lo que en un principio, es muy probable que alguna empresa ofrezca este servicio a sus clientes. Por esta razón se ha decidido implementar un sistema de supervisión que nos permita monitorizar todas o parte de las centralitas que formen el clúster. Hoy en día existen sistemas operativos que permiten instalar directamente un Asterisk junto con una aplicación web que ayuda al mantenimiento de la centralita, en cuanto a configuración se refiere. Este tipo de sistemas pueden ser Trixbox o Asterisk Home, este último desarrollado por la propia empresa Digium. La aplicación web desarrollada en este proyecto fin de carrera pretende monitorizar todas las centralitas que un posible administrador de red quisiera y no limitarse a una única centralita. De este modo, todo el conjunto de este proyecto fin de carrera sería un producto más que interesante para cualquier empresa que quisiera integrarse en las PBX-IP o incluso mejorarla, ya que se le ofrece una red de alta disponibilidad asegurándoles un servicio continuo junto con una aplicación web capaz de monitorizar las centralitas con la intención de ver posibles fallos en el sistema.

Capítulo 4

Memoria técnica del proyecto

A lo largo del siguiente capítulo se le explicará al lector todo el funcionamiento de la red de alta disponibilidad para las centralitas Asterisk, su configuración y explicación detallada de todos los módulos que forman parte de ella. Como no podía ser menos, también se dará una explicación detallada sobre el sistema de supervisión creado, esto es la forma de programarlo, sus diferentes clases Java que componen la aplicación, explicada una a una, y por supuesto cómo funciona la aplicación web.

Para ello, el capítulo se divide en dos apartados. El primero de ellos dará una breve descripción acerca de los módulos que forman parte en esta red de alta disponibilidad, así como un esquema de red que permita al lector visualizar de forma gráfica como quedaría constituida la red a implementar. En el segundo apartado es donde se entrará en detalle en cada uno de los módulos para explicar de forma precisa cómo funciona y cómo debe configurarse (sólo se mostrarán las líneas de configuración más importantes, los ficheros completos se podrán consultar en el ANEXO F). Para el sistema de supervisión, se mostrarán además imágenes de cómo queda la aplicación vista desde un explorador web.

4.1 Esquema general Propuesto

A continuación se muestra un diagrama de bloques de manera que podamos ver de forma gráfica como quedaría nuestra red implementada:

Como se aprecia, entran en juego diferentes módulos para que todo esto funcione, por tanto, se dará una breve introducción de todos ellos para posteriormente profundizar en su uso:

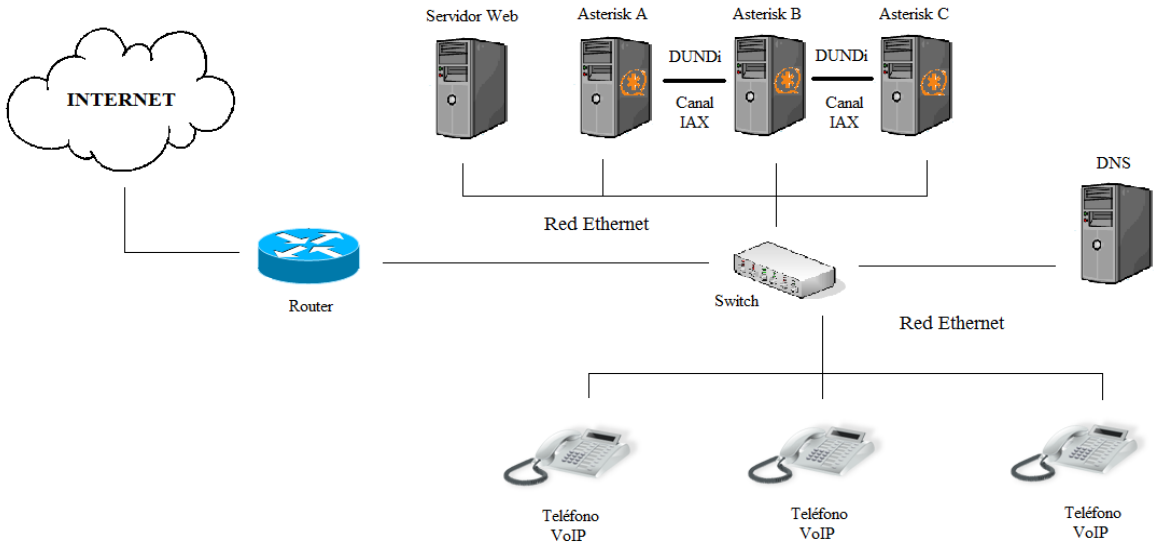

Figura 1: Esquema básico de red

- **Servidor DNS:** Es el encargado de proporcionar una de las direcciones IP de las centralitas Asterisk a todos aquellos terminales que deseen registrarse, así como del balance de carga.

- **PBX Asterisk:** Son las centralitas telefónicas y como tal el núcleo de la red. Éstas serán las encargadas de gestionar los registros, las llamadas en curso…pero todo ello con la ayuda de la base de datos.

- **Base de datos MySQL:** Ésta contendrá la información de los terminales que gestionan las centralitas y del plan de marcado (Dialplan) así como del registro de llamadas. Será aquí donde acuda la centralita para poder validar un terminal o poder realizar una llamada.

- **Protocolo DUNDi:** Protocolo diseñado por el creador de Asterisk. Permitirá que las centralitas se pregunten entre ellas cuando llamamos a una extensión que no está gestionada por nuestra centralita.

- **Servidor Web:** Contendrá la aplicación web para poder monitorizar las centralitas.

Después de esta breve introducción de los diferentes módulos que componen nuestra red, podemos explicar de forma clara y concisa cómo funciona la red de alta disponibilidad:

Un usuario conecta su terminal a la red, con el objetivo de registrarse en una de las centralitas. El terminal antes de enviar su petición de registro a la centralita, tendrá que preguntar al servidor DNS cuál es la dirección IP de ésta. De esta forma, podemos apreciar dos conceptos importantes en este punto; primero observamos que en la configuración de los terminales no indicamos una dirección IP de una de las centralitas como se haría habitualmente, si no que vamos a indicar un nombre de dominio (éste puede ser ficticio puesto que es interno a nuestra organización) que nos mande directamente al servidor DNS; y segundo, será el servidor DNS el encargado de gestionar el uso de las centralitas. Una vez el servidor DNS nos proporcione la dirección IP de una de nuestras centralitas, el terminal enviará la petición de registro a la centralita proporcionada. Cuando ésta obtenga dicha petición, tendrá que consultar en la base de datos si el terminal solicitado puede ser registrado. Si todo es correcto, el terminal quedará registrado en la base de datos y podrá realizar las llamadas pertinentes. Pero llegados a este punto pueden surgir algunas dudas, tales como:

¿Qué ocurre cuando los usuarios son registrados en centralitas diferentes? ¿Cómo puede ser que se llamen si no están gestionados por la misma centralita?

Pues bien, es aquí donde entra en juego el protocolo DUNDi. Cuando un terminal llama a una extensión que no está gestionada por su propia centralita, ésta hará una consulta DUNDi al resto de centralitas para averiguar quién gestiona esa extensión y poder enviar la llamada a su centralita.

Con todo esto, se puede afirmar que la red estaría funcionando correctamente y lo que es más importante, con la garantía de que si una de las centralitas se cae, los

terminales gestionados por esa centralita volverían hacer una consulta al servidor DNS para que éste les proporcione otra de las IP's disponibles y poder registrarse en otra centralita diferente.

Por último, entrará en juego el sistema de supervisión. Éste nos permitirá monitorizar las centralitas del clúster en tiempo real, de forma que nos aporte información acerca de todas ellas. Lo más importante será la inclusión de nuevos usuarios una vez las centralitas ya estén funcionando, de esta forma, podremos ampliar el número de extensiones sin necesidad de recargar Asterisk. Como se verá posteriormente, introducir un nuevo usuario implicará modificar el plan de marcado, dialplan, pero no será ningún problema ya que el sistema de supervisión también permitirá llevar esta tarea en tiempo real y sin necesidad de recargar Asterisk. Otra característica importante será controlar el registro de llamadas, otra aplicación dentro de nuestro sistema de supervisión que va a permitir a un posible administrador conocer las llamadas efectuadas en las distintas centralitas.

Una peculiaridad interesante respecto a las extensiones que se agreguen utilizando el sistema de supervisión, será la necesidad de hacer una réplica de la información suministrada en las diferentes bases de datos que se dispongan, en esta ocasión, una base de datos por centralita Asterisk, ya que la base de datos irá alojada en el mismo equipo que la centralita. El por qué de esta decisión se podrá conocer en el siguiente apartado en el módulo de "Base de datos MYSQL".

4.2 Descripción funcional de cada uno de los módulos desarrollados

A continuación se mostrará de forma más detallada la funcionalidad de cada uno de los módulos que entran en juego en nuestra red.

4.2.1 Terminales

Los terminales serán los encargados de permitirnos realizar las llamadas. Podrán ser teléfonos físicos, conocidos como hardphone, o en su defecto programas informáticos que cumplen la misma función, llamados softphone. La peculiaridad de estos teléfonos es que tienen que tener soporte para DNS SRV, ya que nuestra red funcionará bajo este protocolo y es requisito fundamental para los terminales.

4.2.2 Servidor DNS

En las siguientes líneas se explicará la funcionalidad concreta del servidor DNS en torno al proyecto. Para más información sobre DNS se puede consultar el ANEXO C.

Para la ejecución del proyecto fin de carrera hay que dejar claro que no se hará uso del servidor DNS como la utilidad que comúnmente se le conoce, es decir, para resolución de nombres. En nuestro caso, su uso será DNS SRV. Éste se refiere a uno de los registros de DNS (conocido como SeRVice) que nos permite anunciar los servicios que ofrece nuestro dominio, en nuestro caso SIP.

Como ya se ha comentado en un principio, el servidor DNS será el encargado de proporcionar las direcciones IP's de las centralitas a todos aquellos terminales que deseen registrarse y poder conectarse así a la red. Como es éste quien gestiona las direcciones, se puede conseguir fácilmente un balanceo de carga, es decir, podemos decidir qué centralitas pueden tener mayor protagonismo a la hora de recibir registros, ya sea, por ejemplo, porque tenga mayor capacidad de procesamiento o porque nos interese tener liberada una de las centralitas por consumo de recursos.

Para llevar a cabo la configuración del servidor DNS se usará como software BIND (*Berkeley Internet Name Domain)* comúnmente usado en internet sobre sistemas Unix, en nuestro caso montado sobre una plataforma Debian.

Lo primero de todo será modificar el fichero de configuración para posteriormente configurar el fichero de zona. El archivo de configuración se conoce como named.conf. En él se indican todas las zonas a las cuales dará servicio el servidor DNS, entre otras funciones. A continuación se muestran las líneas de configuración del dominio, los archivos de configuración completos se pueden ver en el ANEXO F.1.1

```
zone "." {
        type hint;
        file "/etc/bind/db.root";
};

zone "redSIP.fic" {
        type master;
        file "/etc/bind/db.redSIP";
        allow-transfer {
                192.168.240.105
        };
};
```

Como se puede apreciar, hay diferenciadas dos zonas. La primera de ellas indica al sistema de nombres de dominio que consulte a los servidores raíz de internet para resolver cualquier nombre que no sea local. Esto permite que el servidor DNS pueda actuar también para la resolución de nombres, aunque no sea el objetivo dentro del proyecto. De esta forma el servidor actúa como servidor de internet.

La segunda zona configurada será el dominio dentro de la red, conocido como redSIP. La sentencia *type* con valor *master* le indica que es el servidor principal de la zona. Esto es importante porque como se verá más adelante, también será necesario configurar un servidor secundario y el valor de dicha sentencia cambiará. A continuación, en *file*, debemos indicarle donde se encuentra nuestro fichero de zona que contendrá las direcciones de nuestras centralitas, entre otras cosas. Por último, la sentencia *allow-transfer ,* especifica las máquinas que pueden copiarse la información de zona de esta máquina, es decir, lo servidores secundarios. Con todo esto, se puede pasar a configurar nuestro fichero de zona.

Un servidor primario para una zona necesita un fichero que contenga todos los registros de recurso asociados a dicha zona. Los datos de cada zona se introducen en un fichero de texto, cuyo nombre se especifica en *named.conf*, en la sentencia *type*, como se vio anteriormente. Este fichero es el conocido como fichero de zona. Para este proyecto, se necesita definir un registro de recurso que es el que hace funcionar todo el entramado de la red de alta disponibilidad, el registro de recurso SRV. Estos registros permiten localizar varios servidores que proporcionen un servicio basado en TCP/IP similar mediante una única operación de consulta DNS. Este registro mantiene una lista de servidores para un tipo de protocolo de transporte y un puerto de servidor conocidos ordenada por preferencia para un nombre de dominio DNS. También cabe mencionar que gracias a este registro es por el cual se puede conseguir un balanceo de carga en las centralitas gracias a uno de sus parámetros. A continuación se van a mostrar las líneas de configuración relevantes a estos registros, el archivo completo se puede ver en el ANEXO F.1.2

```
pbxA IN A 192.168.240.115
pbxB IN A 192.168.240.120

_sip._udp.redSIP.fic. 300 IN SRV 0 0 5060 pbxA.redSIP.fic.
_sip._udp.redSIP.fic. 300 IN SRV 0 1 5060 pbxB.redSIP.fic.
```

Las dos primeras sentencias son registros de tipo A, lo que quiere decir que nos aportan la dirección IP de una máquina, en este caso las dos centralitas, conocidas por el nombre pbxA y pbxB. Las dos últimas sentencias son pertenecientes al registro SRV. Para entender mejor su descripción, se dará una explicación más detallada de esta sentencia.

Los objetivos de cada uno de los campos especializados que se usan en el registro de recursos SRV son los siguientes:

- **Servicio** (_sip) un nombre simbólico para el servicio deseado. Para los servicios conocidos, se define en el documento RFC 1700 un nombre simbólico universal reservado como "_telnet" o "_smtp". Si no se define en el documento RFC 1700 un nombre de servicio conocido, en su lugar se puede utilizar un nombre preferido por el usuario. Si el RFC 1700 asigna un nombre para un servicio indicado en este campo, el nombre definido en RFC es el único nombre que se puede utilizar. Sólo se puede nombrar localmente los servicios definidos de esta forma.

- **Protocolo** (._udp) indica el tipo de protocolo de transporte. Normalmente, es TCP o UDP, aunque se puede utilizar cualquier protocolo de transporte nombrado en RFC 1700.

- **Nombre** (redSIP.fic.) el nombre de dominio DNS al que se refiere este registro de recursos. El registro de recursos SRV es diferente con respecto a otros tipos de registro DNS en que no se utiliza para realizar búsquedas o consultas.

- **Prioridad** (0) establece la preferencia para un host especificado en el campo *destino*. Los clientes DNS que consultan registros de recursos SRV intentan

ponerse en contacto con el primer host alcanzable de la preferencia con el número más bajo de la lista. Aunque los hosts de destino tengan el mismo valor de preferencia declarado, se pueden probar en orden aleatorio. El intervalo de los valores de preferencia es de 0 a 65535.

- **Peso** (0) se puede utilizar además de preferencia para proporcionar un mecanismo de equilibrio de carga donde se especifican varios servidores en el campo **destino** y se establecen todos en el mismo nivel de preferencia. Cuando seleccione un host servidor de destino entre los de igual preferencia, este valor se podrá utilizar para establecer un nivel agregado de preferencia que se puede usar para determinar el orden exacto o el equilibrio de selección para los hosts de destino utilizados en una consulta SRV. Cuando se usa un valor diferente de cero, se prueban los servidores de igual preferencia en proporción al peso de este valor. El intervalo de valores es de 1 a 65535. Si no se requiere el equilibrio de carga, utilice el valor **0** en este campo para facilitar la lectura del registro.

- **Puerto** (5060) el puerto servidor en el host de destino que proporciona el servicio indicado en el campo servicio. El intervalo de números de puerto es de 0 a 65535, aunque el número es, a menudo, un número de puerto de servicio asignado conocido, según se especifica en RFC 1700. Los puertos sin asignar se pueden utilizar según sea necesario.

- **Destino** (*target*) especifica el nombre de dominio DNS del host que proporciona el tipo de servicio que se solicita. Para cada nombre de host utilizado, se requiere su correspondiente registro de recursos de direcciones de host (A) en el espacio de nombres DNS. Se puede utilizar un único punto (**.**) en este campo para indicar de forma autorizada que el servicio solicitado especificado en este registro de recursos SRV no está disponible en este nombre de dominio DNS.

Si se observa en la primera sentencia de los registros SRV, el campo peso tiene como valor ‘1’, en vez de ‘0’. Esto permite que la primera centralita tenga preferencia sobre la segunda, de forma que reciba más peticiones. Esto es lo que se conoce como balanceo de carga.

Para terminar con este módulo, hay que hablar de la necesidad de instalar un servidor DNS secundario. Como se sabe, existen dos tipos de DNS, primarios o maestros y secundarios o esclavos. Estos últimos son necesarios para dar soporte cuando el primario esté fuera de servicio. Son de vital importancia ya que cumplen una tarea exactamente igual que la de un servidor primario cuando éste deja de funcionar. Por ese modo, si queremos que la red de alta disponibilidad que se implemente tenga tal efecto, será necesario instalar un segundo servidor DNS por si el primero de ellos queda inutilizado, y con ello, la imposibilidad del registro de los terminales, por lo que los usuarios no tendrían servicio. Por lo general, los servidores DNS suelen estar fuera de la red a la que aportan servicio al igual que servidores primarios y secundarios están en distintas redes, ya que si la red del servidor primario se cae, el secundario puede seguir dando servicio. Para este proyecto esta condición no es necesaria, ya que si la red se cae, los terminales no podrían realizar sus llamadas, aunque el servidor secundario se encontrase en una red diferente.

Con todo esto, la configuración del servidor secundario cambia en la definición de la zona en el archivo named.conf, quedando de la siguiente forma:

```
zone “redSIP.fic” {
        type slave;
        file “/etc/bind/db
        master {
                192.168.240.100
        };
}
```

Como vemos, la sentencia *type* indica que será servidor secundario de nuestra zona redSIP.fic. La sentencia *masters* indica la dirección IP del servidor primario de la zona. Por último, un servidor que actúa de secundario para una zona, no necesita configurar el fichero de zona puesto que lo obtiene de forma periódica del primario. Esta descarga se realiza automáticamente. Además, se puede configurar un servidor primario para que notifique a los secundarios cada vez que se produzca algún cambio en la base de datos de la zona (en terminología DNS esto se conoce como *notify*). Este mecanismo permite una propagación muy rápida de las actualizaciones. Es por esto que la sentencia *file* indica donde almacenar el fichero de zona transferido del servidor primario, y por tanto, donde puede consultarlo.

4.2.3 Base de datos MySQL

Este módulo es el referente al sistema de Asterisk basado en RealTime. La tecnología RealTime nos permite introducir cambios en la configuración de Asterisk sin necesidad de recargarlo para que actualice los cambios. Esto se debe, como ya se comentó, a que ciertos módulos de configuración de Asterisk pueden ser configurados en una base de datos. De esta forma, si existe la necesidad de introducir un nuevo usuario, basta con poner la información necesaria en la tabla de la base de datos que gestione ese cometido. Así, la información es actualizada en tiempo real y sin necesidad de recargar Asterisk, siendo ésta la principal ventaja. Además, para nuestro trabajo, al realizar un sistema de supervisión de las centralitas nos facilita mucho el trabajo poder obtener la información de la base de datos.

Para llevar a cabo RealTime, se ha obtado por utilizar una base de datos MySQL.

Es importante en este punto aclarar uno de los problemas planteados para poder crear la red de alta disponibilidad en lo que a diseño se refiere. Existen diferentes posibilidades para montar las bases de datos dentro de la red de Asterisk. A continuación se exponen todas ellas.

En un primer lugar, lo más intuitivo y apropiado parece montar una base de datos en un equipo independiente a las centralitas y que todas éstas accedan a la información necesaria para poder realizar su función en RealTime. El problema de esto es que si la base de datos se cae o queda inutilizada, el sistema telefónico dejaría de funcionar, ya que aunque los usuarios estuvieran registrados, la reglas de marcado para poder llamar de una extensión a otra también residen en la base de datos, por lo que

sería imposible llevar a cabo por parte de las centralitas las posibles llamadas. Debido a esto, nuestra red de alta disponibilidad tendría una falla importante de servicio, por lo que esta solución de diseño no es viable.

Una segunda solución parece inmediata tras leer la primera opción de diseño. Al igual que los servidores DNS, se podría implementar un segundo servidor con una base de datos replicada de la principal, de forma que si la primera queda fuera de servicio, ésta pasaría a ser utilizada por parte de las centralitas. Sin duda esta opción parece la más viable y la más recomendada, si no fuera por culpa de la configuración de Asterisk. El software de Asterisk nos permite trabajar en RealTime y con ello hacer uso de una base de datos externa a la centralita, pero no permite indicarle otra posible dirección de respaldo por si el equipo donde reside la base de datos queda inutilizado. Sabiendo esto, esta solución queda igualmente descartada.

Por último, sólo quedaría una solución posible que como cabe de esperar, es instalar las bases de datos en el mismo equipo donde reside la centralita Asterisk. Sin duda, esta opción parece la más pesada pero sin duda la más eficaz, si tenemos en cuenta que se pretende conseguir una red de alta disponibilidad. Además, hay que tener en cuenta que se va a realizar una aplicación web que nos permita monitorizar las centralitas, por ejemplo, entre otras cosas, los usuarios registrados en cada centralita. Esto sólo se puede conseguir si asignamos a cada centralita su propia base de datos. También puede parecer engorroso realizar una réplica de las bases de datos por centralitas formen el clúster, pero para eso se programará el sistema de supervisión de forma que éste haga la réplica en todas ellas.

Sabiendo el diseño que tendrá nuestra red en cuanto a las bases de datos, se puede continuar con la configuración de ésta.

Los módulos a desarrollar en RealTime serán los relacionados con la creación de extensiones y registro de usuarios, pertenecientes al archivo sip.conf, al dialplan, en este caso perteneciente al archivo extensions.conf y al registro de llamadas, que utiliza el archivo cdr_mysql.conf. Por tanto, habrá que crear una base de datos que contenga tres tablas, como a continuación se explica:

Una vez tengamos instalado MySQL, podemos acceder a éste de la siguiente forma:

mysql –u root –p
Enter password:

Donde root es el usuario y –p servirá para poder introducir la contraseña del usuario.

Una vez dentro de la consola de MySQL, lo primero de todo será crear un usuario con todos los privilegios, siendo éste el encargado de gestionar la base de datos de Asterisk. Este usuario será con el que se conecte a la base de datos cuando Asterisk esté en ejecución y necesite acceder a ésta. Esto se verá más adelante en el módulo PBX Asterisk en la configuración del archivo res_mysql.conf. Para crear un usuario con todos los privilegios basta con escribir la siguiente sentencia:

mysql> GRANT ALL PRIVILEGES ON *.* TO userAsterisk@'%' IDENTIFIED BY '030704' WITH GRANT OPTION;

Para comprobar que está habilitado el nuevo usuario, podemos salir de la consola de MySQL, con el comando *exit*, y volver a ingresar con el nuevo usuario creado.

Ahora se debe crear nuestra base de datos, que tendrá por nombre "asterisk", por lo que ejecutamos la siguiente sentencia:

mysql> CREATE DATABASE asterisk;

Nuevamente, para comprobar que todo se ejecutó de forma correcta, se puede comprobar con el comando *show databases;* si la base de datos existe. De ser así, ingresamos a ésta con el comando *use asterisk;*. Como consejo, se puede decir que una vez tengamos la base de datos creada, podemos acceder a mysql con la siguiente sentencia: *mysql –u usuario –p asterisk* , de esta forma, accedemos directamente a nuestra base de datos. Una vez dentro de la base de datos asterisk, que será la que se utilice para trabajar en RealTime, hay que crear las tablas necesarias para su correcto funcionamiento.

Como ya se ha comentado, se necesitan crear tres tablas, una para las extensiones y registros, otra para el dialplan y una última para el registro de llamadas. A continuación se muestra como deben ser dichas tablas:

La primera de ellas será la relacionada a las extensiones. Se deberá crear con la siguiente sentencia y con los campos mostrados:

```
mysql> CREATE TABLE usuariosSIP (id int(20) NOT NULL auto_increment,
name varchar(80) NOT NULL default '',
type varchar(45) NOT NULL default 'friend',
ipaddr varchar(15) NOT NULL default '',
port int(6) NOT NULL default '0',
host varchar(31) NOT NULL default 'dynamic',
secret varchar(80) NOT NULL default '',
username varchar(80) default '',
context varchar(80) NOT NULL default 'internal',
disallow varchar(45) NOT NULL default 'all',
allow varchar(45) NOT NULL default 'g729,gsm,ulaw,alaw,ilbc',
regseconds int(11) NOT NULL default '0',
qualify varchar(45) NOT NULL default 'yes',
canreinvite varchar(45) NOT NULL default 'no',
nat varchar(45) NOT NULL default 'no',
cancallforward varchar(45) NOT NULL default 'yes',
PRIMARY KEY (id),
UNIQUE KEY name (name)
) ENGINE=MyISAM;
```

Como se puede observar, hay un número elevado de campos en la tabla, por lo que se explicarán los más importantes de ésta:

- **name:** Nombre que le damos a la extensión. Puede ser un número, nombre o combiación de ambas.

- **type:** Se usa para autenticar llamadas entrantes, salientes o ambas**.** Para este trabajo se ha definido una extensión type=friend que permite realizar y recibir llamadas.

- **ipaddr:** Dirección IP de la extensión registrada. Éste campo será muy útil para saber qué extensiones están registradas.

- **port :** Puerto de la extension registrada. Al igual que ipaddr, sólo se rellena cuando una extensión se registra contra la centralita.

- **host:** Da la opción al terminal de poder conectar desde cualquier dirección IP o una fija. En este caso, por defecto está configurado como host=dynamic, por lo que podrá conectarse desde cualquier IP, pero siempre será necesario identificar el campo secret.

- **secret:** Es la contraseña usada para la autenticación entre la centralita y el terminal.

- **context:** Indica el contexto donde están las instrucciones de marcado para esa extensión.

- **disallow:** Permite desabilitar codecs

- **allow:** Permite habilitar un codec. Pueden ponerse varios en un mismo usuario

- **regseconds:** Tiempo de registro

- **qualify:** Monitoriza la latencia entre el servidor Asterisk y el teléfono para determinar cuando el dispositivo puede ser alcanzado. En este caso Asterisk considera por defecto que un dispositivo está presente si su latencia es menor de 2000 ms. Por defecto se configura como qualify=yes.

- **canreinvite:** Permite establecer la comunicación entre dos extensiones de forma que los paquetes RTP de audio podrían ser enviados extremo a extremo sin pasar por el servidor Asterisk. Esto, normalmente, no suele ser conveniente en casos en los que haya NAT en alguno de los clientes. (NAT=yes). Por defecto está configurado como canreinvite=yes, ya que nuestros terminales no estarán detrás de un NAT.

- **nat:** Indica si el dispositivo está detrás de un NAT. Para este trabajo no fuerza dicha situación por lo que estará como nat=no por defecto.

Ahora se creará la tabla correspondiente al dialplan. Como se observará a continuación, esta tabla cuenta con menos campos que la anterior pero siendo todos ellos de vital importancia, quedando de la siguiente forma:

```
mysql> CREATE TABLE dialplan (
 id int(11) NOT NULL auto_increment,
 context varchar(20) NOT NULL default '',
 exten varchar(20) NOT NULL default '',
 priority tinyint(4) NOT NULL default '',
 app varchar(20) NOT NULL default '',
 appdata varchar(128) NOT NULL default '',
 PRIMARY KEY  (`context`,`exten`,`priority`),
 KEY `id` (`id`)
) ENGINE=MyISAM;
```

Como en la tabla anterior, se explicará la función de los cinco campos relevantes a las extensiones:

- **context:** Es el nombre del contexto donde se encuentra la información de marcado de las extensiones. Corresponde al contexto que se indicó en la tabla usuariosSIP.

- **exten:** Corresponde con la extensión a la que se va aplicar el dialplan.

- **priority:** Orden en que se ejecutan las sentencias de marcado para la extensión. La de mayor prioridad será la número 1.

- **app:** Tipo de aplicación que se va a ejecutar en esta sentencia, pudiendo ser, entre otras, de llamada (Dial), colgado (Hangup)…

- **appdata:** Datos necesarios para la ejecución de la acción a tratar.

Para comprenderlo mejor, se muestra un ejemplo de sentencia que iría alojado en el archivo extensions.conf:

```
[ejemplo]
exten=> 100,1,Dial(SIP/100,10)
exten=> 100,2,Hangup
```

Si lo comparamos con los campos de la tabla, podemos observar que esta extensión pertenece al contexto "ejemplo" siendo las dos sentencias pertenecientes a la extensión número 100, por lo que en la tabla escribiríamos context=ejemplo y exten=100. Tiene como mayor prioridad y por tanto primera sentencia a ejecutar la que está configurada con el valor 1, por tanto, priority=1 (siempre se suelen poner en orden de ejecución) siendo ésta una sentencia de llamada, es decir, app=Dial. Lo que se encuentra entre paréntesis sería el equivalente a appdata, sentencia necesaria para que en este ejemplo se llame a la extensión 100. La segunda sentencia corresponde de fin de llamada, siendo app=Hangup y appdata vacío puesto que no requiere ninguna información adicional.

Por último, se crea la última tabla necesaria para nuestro trabajo en RealTime, que corresponde con el registro de llamadas, o como se conoce en Asterisk, CDR.

```
CREATE TABLE cdr (id int(20) NOT NULL auto_increment,
calldate datetime NOT NULL default '0000-00-00 00:00:00',
clid varchar(80) NOT NULL default '' ,
src varchar(80) NOT NULL default  '',
dst varchar(80) NOT NULL default '' ,
dcontext varchar(80) NOT NULL default '' ,
channel varchar(80) NOT NULL default  '',
dstchannel varchar(80) NOT NULL default  '',
lastapp varchar(80) NOT NULL default '' ,
lastdata varchar(80) NOT NULL default '' ,
duration int(11) NOT NULL default '0',
billsec int(11) NOT NULL default '0',
disposition varchar(45) NOT NULL default  '',
amaflags int(11) NOT NULL default '0',
accountcode varchar(20) NOT NULL default  '',
uniqueid varchar(32) NOT NULL default '' ,
userfield varchar(255) NOT NULL default '',
PRIMARY KEY (id),
) ENGINE=MyISAM;
```

A continuación se muestran los campos más significativos de esta última tabla:

- **calldate:** Día y hora de la llamada.
- **src:** Extensión que realiza la llamada.
- **dst:** Extensión a la cual va dirigida la llamada.
- **dcontext:** Contexto al que pertenecen las extensiones.
- **channel:** Canal de comunicación.
- **dstchannel:** Canal de comunicación en el destino.
- **lastapp:** Última aplicación ejecutada.
- **lastdata:** Los datos de la última aplicación ejecutada.
- **duration:** Duración de la llamada.
- **disposition:** Estado de la llamada.

Para terminar, hay que tener en cuenta que el acceso a las bases de datos por parte del sistema de supervisión se hará desde una dirección remota, es decir, no se ejecutará sobre localhost. Por defecto, mysql no permite conexiones remotas a la máquina donde se encuentra instalado, por lo que será necesario deshabilitar dicha opción. De este modo, se debe ir al fichero de configuración de mysql, my.cnf (en instalaciones Debian se encuentra en /etc/mysql/my.cnf). Una vez allí, hay que deshabilitar la siguiente sentencia, poniendo una almohadilla delante del inicio de ésta:

Con todo esto, ya tenemos nuestra base de datos lista para ser utilizada por parte de las centralitas Asterisk como de nuestro sistema de supervisión.

4.2.4 Protocolo DUNDi

Distributed Universal Number Discovery (DUNDi), es un protocolo diseñado por el creador de Asterisk Mark Spencer. Su finalidad principal era crear sistemas distribuidos de Asterisk por medio de una red peer-to-peer, en la cual no existen roles fijos de clientes y servidores, sino que pueden asumir uno u otro rol. De esta forma, se pretendía tener centralitas Asterisk es distintos puntos de la red y conseguir que los usuarios de unas y otras se llamasen entre sí. Explicado con un caso práctico para mayor entendimiento, podríamos tener una empresa con distintas sedes, ya sea en Madrid, Barcelona y Sevilla, por ejemplo. La centralita de Madrid gestiona las extensiones de la 100 a la 199, la de Barcelona de la 200 a la 299 y la de Sevilla de la 300 a la 399. Por tanto, pretendemos que usuarios de distintas sedes puedan llamarse entre sí. Cuando un usuario de Madrid quisiera llamar a una extensión fuera del rango de las que gestiona su centralita, la llamada sería rechazada, por tanto, es aquí donde entra en juego el protocolo DUNDi. Éste será el encargado de publicar las extensiones que gestiona la centralita. De esta forma, si un usuario de la sede de Madrid, con extensión 101, desea llamar a otro usuario de Barcelona con extensión 201, la centralita de Madrid hará una consulta DUNDi tanto a Barcelona como a Sevilla para saber cuál de las dos centralitas gestiona esa extensión. En este caso, la centralita de Barcelona contestaría a la de Madrid para poder establecer la llamada. Principalmente, este es el objetivo de DUNDi, pero en nuestro caso, lo usaremos para establecer un clúster de Asterisk como ya se ha comentado a lo largo del documento.

Para este proyecto, no se tendrán centralitas en distintas redes, sino todas en la misma red y todas gestionando las mismas extensiones. De esta forma, se utilizará DUNDi para establecer la comunicación entre ellas y puedan anunciarse, en este caso, no las extensiones que gestionan, ya que son las mismas, si no las extensiones que están registradas en el momento de la llamada. Esto se debe a que como ya se ha comentado, vamos hacer uso de DNS SRV, por tanto, a la hora de asignarnos dirección IP, puede asignar diferentes direcciones por lo que los usuarios se encontrarán en centralitas distintas. Cuando un usuario llame a una extensión que gestione su misma centralita, no habrá problema ya que la llamada se conoce como “llamada local” y sería directa. En caso de pertenecer a una centralita diferente, será cuando se aplique la consulta DUNDi al resto de centralitas para saber en cuál se encuentra y poder gestionar la llamada entre ambas. Para conocer como se debe configurar la centralita para que se lleve a cabo todo este proceso, podremos verlo posteriormente en el módulo PBX Asterisk en los archivos de configuración dundi.conf e iax.conf.

4.2.5 PBX Asterisk

Como ya sabemos, Asterisk es una centralita de telefonía que permite el uso de la tecnología VoIP, siendo Asterisk la plataforma actual más extendida en este campo. En este apartado se explicará cómo deben ser configuradas las centralitas para su correcto funcionamiento. Cabe destacar que ahora sólo se hará referencia a las líneas de configuración más importantes para que la red de centralitas funcione de la forma

esperada, en el ANEXO F.3 se podrá apreciar los siguientes ficheros de configuración en su totalidad.

4.2.5.1 Archivo extconfig.conf

En el siguiente archivo será donde se indiquen los módulos que deseamos cargar en tiempo real, RealTime, haciendo uso de la base de datos. Para este proyecto, serán los usuarios y los peers SIP y el dialplan, por lo que tendrá que tener las siguientes líneas:

```
sipusers => mysql, asterisk,usuariosSIP
sippeers => mysql,asterisk,usuariosSIP
extensions => mysql,asterisk,dialplan
```

La comprensión de estas líneas es sencilla; por un lado, indican el tipo de base de datos que se va a utilizar, en este caso será MySQL; en segundo lugar hay que indicar el nombre de la base de datos, que será asterisk; y por último, el nombre de la tabla que contendrá la información pertinente.

4.2.5.2 Archivo res_mysql.conf

Siguiendo con la configuración de RealTime, en este archivo será donde se indique cómo acceder a la base de datos. Bastará con introducir el nombre de la base de datos, un usuario capaz de gestionarla con su correspondiente contraseña, la dirección o el nombre de la máquina donde reside la base de datos y una de las sentencias más importantes, indicar donde se encuentra nuestro conector con la base de datos. Si esta última sentencia no es correcta, podremos conectar con la base de datos pero no podremos modificar las tablas que se necesiten. Hay que decir que la ubicación del sock con la base de datos cambia según la distribución utilizada. Como en este caso será Debian, éste se encontrará en /var/run/mysqld/mysqld.sock.

4.2.5.3 Archivo dundi.conf

Es aquí donde debemos aplicar la configuración para que las centralitas puedan establecer las llamadas a las extensiones que no están registradas en ella misma.

Lo primero de todo será identificar las centralitas para que se conozcan en la red DUNDi que se genera, pudiendo realizarlo con la sentencia entityid, que tendrá un valor igual a la dirección MAC de nuestro interfaz de red que utilicemos para conectarnos, teniendo así la siguiente forma:

```
entityid=AA:BB:CC:DD:EE:FF
```

Se puede poner información acerca del servidor, pero al no ser relevante se podrá observar dicha configuración en el ANEXO donde se muestran los ficheros de configuración en su totalidad.

Como ya sabemos, DUNDi servirá para preguntar al resto de centralitas por aquellas extensiones que no tienen registradas, por lo que tendremos que tener una sentencia del siguiente tipo dentro de la sección conocida como mappings:

[mappings]

```
troncalIAX=>registrosSIP,0,IAX2,dundi:${SECRET}@{$IPADDRESS}/{$EXTENSION}/options
```

A continuación se explica el significado de cada una de las partes de esta sentencia:

- **troncalIAX**: Este es el nombre del recurso. Sólo se usará para buscar extensiones entre las distintas centralitas. Es obvio que este nombre deberá ser el mismo en todas las centralitas, ya que, como veremos a continuación, se creará un canal entre las distintas centralitas para pasar su información DUNDi.

- **registrosSIP**: Es el contexto donde están definidas las extensiones que han sido registradas. Esto quiere decir que cuando pregunten por una extensión que está registrada en dicho contexto, podrá contestar a la petición y propiciar por tanto la llamada

- **0**: Es el peso de la respuesta. Cuanto menor sea mayor prioridad. Es útil en otro tipo de recursos. A la hora de buscar rutas de menor coste para las llamadas, si estamos seguros de que es la mejor, pondremos 0. Si nuestra ruta es buena, pero las hay mejor, pondremos un valor mayor.

- **IAX2**: Es el tipo de canal que vamos a utilizar entre las centralitas. Puede ser SIP, H323 o cualquier otro.

- **dundi:${SECRET}/1.2.3.4/${NUMBER}**: En una cadena de llamada IAX2, "dundi" será el usuario (podemos decir también que será el nombre del canal de comunicación entre las centralitas), ${SECRET} se sustituirá por la contraseña a utilizar. Esto se debe a que cuando definimos *peers* IAX2, hay un parámetro *dbsecret*, que indica que la contraseña está almacenada en AstDB, por ejemplo en dundi/secret. DUNDi genera esas contraseñas, y las va rotando cada cierto tiempo. Permite que los diferentes nodos puedan contactar con nosotros (y llamar a las extensiones que hemos publicado), pero sin darles una contraseña de acceso fija. 1.2.3.4 nuestra IP, y ${NUMBER} se sustituirá por el número de la consulta.

Posteriormente habrá que definir la información pertinente a las centralitas que formen el clúster de Asterisk. Esto se hará con los entityid de las centralitas y su información correspondiente, por lo que existirá una sentencia similar a la que sigue por cada centralita del clúster:

[AA:BB:CC:DD:EE:FF]
model=symmetric
host=1.2.3.4
inkey= claveDUNDi
outkey= claveDUNDi

include=troncalIAX
permit=troncalIAX
qualify=yes
order=primary

La información entre corchetes corresponde con el entityid de la centralita remota al igual que la dirección IP ubicada en la sentencia host. Las comunicaciones entre nodos van encriptadas usando clave pública/privada. Deberemos generarlas previamente y almacenarlas en /var/lib/asterisk/keys. Para generar dichas claves, basta con ubicarnos en la carpeta /var/lib/asterisk/keys/ y ejecutar la siguiente sentencia:

astgenkey –n claveDUNDi

Posteriormente, debemos enviar estas dos claves (genera la pública, con extensión .pub y la privada, con extensión .key) al resto de centralitas y a la misma carpeta. El parámetro *inkey* indica la clave a usar en las consultas que nos realicen el resto de centralitas, y *outkey* la que emplearemos nosotros cuando enviemos nuestras consultas. El parámetro *include* indica para qué recursos usaremos este peer, y *permit* los recursos para los que aceptaremos consultas. Como vemos, corresponde con el que definimos en la sección mappings anteriormente.

4.2.5.4 Archivo sip.conf

En condiciones normales, éste es el fichero donde se configuran los usuarios sip, es decir, las extensiones que gestionaría la centralita, pero en este caso, se hará en RealTime, por lo que dicha configuración irá alojada en la tabla de la base de datos que contenga dicho cometido. Aún así, no hay que pasar ciertos puntos interesantes de configuración en este fichero, ya que éste sirve para configurar todo lo relacionado con el protocolo SIP. Es por ello que hay que tener en cuenta algunos detalles importantes a la hora de configurar el fichero:

Se debe indicar la configuración basada en DNS SRV, ya que es parte fundamental de nuestro trabajo, por lo que deberá existir una sentencia igual a:
srvlookup=yes

Por otro lado, se va a utilizar DUNDi, pues bien, como el objetivo será anunciar sólo los usuarios registrados en nuestra centralita, se creará la siguiente sentencia.
sipregistrations=registrosSIP

Esta sentencia creará un contexto con nombre registrosSIP y creará una entrada de tipo Noop() por cada extensión que se registre que servirá principalmente para anunciar en nuestra red DUNDi que ese usuario está registrado en la centralita y que por tanto puede gestionar la llamada. De igual forma, cuando dicho usuario se desconecte, se eliminará dicha entrada del contexto.

4.2.5.5 Archivo iax.conf

Si se recuerda la configuración del archivo dundi.conf, vemos que tenía definido un canal IAX2 con un nombre de usuario llamado dundi, pues bien, es en este archivo donde hay que configurar dicho usuario. En verdad, lo que se está configurando es una troncal de tipo IAX para poder establecer las llamadas entre las distintas centralitas. Dicho esto, el archivo tendrá la forma siguiente:

```
[dundi]
type=user
context=llamada_DUNDi
dbsecret=dundi/secret
disallow=all
allow=ulaw
allow=alaw
allow=gsm
```

Se puede destacar que la sentencia context= llamada_DUNDi indica el contexto al que irán las llamadas entrantes de otras centralitas para ser cursada, perteneciente al archivo extensios.conf, que más tarde se expone.

Es de gran utilidad facilitar cierta información acerca de la configuración de este archivo. Como ya se ha comentado, existen diferentes módulos de Asterisk que permiten se cargados en RealTime, siendo éste uno de ellos, pero como se observa, este fichero no ha sido cargado de tal forma. Esto se debe a que este fichero va en relación con la configuración DUNDi, y como también se ha comentado, DUNDi no permite ser cargado en RealTime, por lo que no es necesario que el archivo iax.conf sea gestionado por nuestro sistema de supervisión. En el Capítulo 6.-Análisis de Futuro, se explicará una posible mejora en relación a este problema expuesto.

4.2.5.6 Archivo extensions.conf

El archivo extensions.conf es el más importante de Asterisk y tiene como misión principal definir el dialplan o plan de marcado que seguirá la centralita para cada contexto y por tanto para cada usuario. Nuevamente, recordar que este archivo está configurado en RealTime, pero tiene una particularidad respecto al archivo sip.conf. En esta ocasión, van a existir sentencias fijas que sí deben encontrarse en el archivo y no en la tabla de la base de datos donde se encuentre el plan de marcado para los contextos. Explicado de una forma más concisa y haciendo relevancia al trabajo llevado en el proyecto, hay que saber que únicamente tendremos un contexto de marcado para las extensiones locales, eso quiere decir que no se harán llamadas fuera de nuestra red, por tanto, sólo el contexto que contenga la sentencia para llamar a dichas extensiones será aquél que resida en la base de datos. Para comprenderlo mejor, a continuación se muestran las líneas de configuración del fichero para el dialplan:

```
[internal]
include => busqueda_DUNDi
```

```
include => exten_SIP
[busqueda_DUNDi]
switch => DUNDi/troncalIAX

[exten_SIP]
switch => RealTime/dialplan@

[llamada_DUNDi]
exten => _2XX,1,Goto(exten_SIP,${EXTEN},1)
```

Como vemos en las líneas anteriores, distinguimos cuatro contextos diferentes, siendo éstos:

internal
busqueda_DUNDi
exten_SIP
llamada_DUNDi

El contexto internal es el contexto por defecto configurado en las extensiones. Éste es el primero al que accede la centralita cuando quiere cursar una llamada. Como vemos, contiene dos sentencias include, lo que quiere decir que acudirá a los contextos señalados por ambas sentencias. En primer lugar, hará una consulta DUNDi para saber si la extensión a la que se llama se encuentra en una centralita remota. Esta consulta se hace desde el contexto búsqueda_DUNDi, que es el encargado de ejecutar la sentencia de búsqueda DUNDi. Si efectivamente la extensión se encuentra en una centralita remota cursará la llamada vía DUNDi; de no ser así, ejecutará el contexto exten_SIP, puesto que la llamada será interna a la centralita. En caso de que la llamada se realice vía DUNDi, es aquí donde entra en juego el contexto llamada_DUNDi. Este contexto es al que accede la centralita cuando se recibe una llamada por DUNDi. Cuando se establece este tipo de llamada, se crea un canal IAX entre ambas centralitas, enviando la llamada al usuario iax configurado en el archivo iax.conf, siendo ahí donde se le indicaba que las llamadas recibidas sean cursadas por el contexto llamada_DUNDi. Así, la llamada llega a dicho contexto, teniendo como sentencia de ejecución la indicada arriba, siendo ésta:

```
exten => _2XX,1,Goto(exten_SIP,${EXTEN},1)
```

Esta sentencia lo único que pretende es enviar la llamada al contexto exten_SIP, a la extensión indicada por ${EXTEN}, que es la extensión a la que se dirige la llamada, y la prioridad a la que debe acceder. Si observamos la línea de ejecución de este último contexto, indica que la información de marcado se encuentra configurada en RealTime, por lo que accederá a la tabla dialplan en busca de la información de marcado.

Por último, comentar que cuando la llamada es local, es decir, entre extensiones gestionadas por la misma centralita, el curso de la llamada es el mismo que el último realizado en una llamada DUNDi, es decir, después de comprobar que la extensión no se encuentra en una centralita remota, pasa directamente al contexto exten_SIP, como

ya se comentó anteriormente, y accede a la base de datos para comprobar las reglas de marcado de la extensión destino.

4.2.5.7 Archivo cdr_mysql.conf

Por último, el archivo cdr_mysql.conf será el encargado de almacenar el registro de llamadas en la base de datos. Gracias a esto, posteriormente en nuestra aplicación web que monitorice las centralitas, nos permitirá ver las llamadas realizadas en cada una de las centralitas del clúster.

4.2.6 Servidor web

Comienza aquí el último módulo del proyecto y el más innovador a lo que puesta en el mercado se refiere. El servidor web será el encargado de alojar el sistema de supervisión que permite monitorizar las centralitas del clúster y trabajar sobre sus respectivas bases de datos. Como ya se ha comentado, la aplicación web está programada con tecnología JAVA, en concreto haciendo uso de Servlets y páginas JSP. Esto implica que hay que utilizar un servidor capaz de soportar dicha tecnología, por lo que se ha decidido instalar un servidor web TOMCAT v6.18. Para más información acerca de TOMCAT, como instalación, estructura, etc, se puede consultar el ANEXO E.

En las siguientes líneas se pretende dar a conocer al lector la estructura concreta llevada a cabo para realizar el sistema de supervisión. En un primer lugar, se dará una explicación detallada de las clases JAVA creadas para posteriormente mostrar un documento creado con JAVADOC que permite mostrar nuestra aplicación como un API de JAVA.

4.2.6.1 Sistema de supervisión

Hay que diferenciar la funcionalidad de los Servlets respecto a los JSP. Una aplicación web realiza tareas de procesado y presentación, de este modo, los Servlets son adecuados para el procesado de la información y las páginas JSP para la presentación. De este modo, cuando el servidor recibe una petición, los parámetros se procesan de modo que:

- El acceso a las bases de datos lo realizan los Servlets.
- La lógica de la aplicación la controlan los Servlets.
- La presentación de la información requerida la harán los JSP.

Por tanto, supongamos que el cliente envía una petición HTTP al servidor: en primer lugar, el Servlet procesa la petición y si es necesario conectará con la base de datos; en segundo lugar el Servlet redirige la petición a un JSP o a otro Servlet (comúnmente a otro JSP si se desea mostrar la información); en tercer y último lugar, el

JSP lee los parámetros y formatea la información de forma que sea visualizada por el usuario.

Una vez se sabe cómo funciona la lógica de la aplicación haciendo uso de Servlets y JSP, se explicará cómo se deben crear los Servlets. En este caso, se explicará la forma utilizada en el proyecto.

Los Servlets programados tendrán la capacidad de manejar peticiones GET y POST. Las peticiones GET son peticiones hechas por el navegador cuando el usuario teclea una URL en la línea de direcciones, sigue un enlace desde una página Web, o rellena un formulario que no especifica un METHOD (atributo del tag FORM en programación HTML). Las peticiones POST son generadas cuando alguien crea un formulario HTML que especifica METHOD="POST". Para ser un Servlet, la clase debe extender de HttpServlet y escribir el método doGet() o doPost() (o ambos), dependiendo de si los datos están siendo enviados mediante GET o POST. Estos métodos toman dos argumentos: un HttpServletRequest y un HttpServletResponse.

El HttpServletRequest tiene métodos que nos permiten encontrar información entrante como datos de un FORM, cabeceras de petición HTTP, o parámetros que residen en la propia URL (se diferencian porque van seguidos de un '?' en la URL). El HttpServletResponse tiene métodos que nos permiten especificar líneas de respuesta HTTP (200, 404, etc.), cabeceras de respuesta (Content-Type, Set-Cookie, etc.), pero para nuestra aplicación no haremos uso de él, aunque si debe aparecer puesto que los métodos doGet() y doPost() necesitan ambos parámetros para su construcción. Ambos métodos lanzan dos excepciones, ServletException e IOException, por eso es necesario incluirlas en la declaración. También es necesario importar las clases de los paquetes java.io (para PrintWriter, etc.), javax.servlet (para HttpServlet, etc.), y javax.servlet.http (para HttpServletRequest y HttpServletResponse).

Ahora se dispone de una visión global acerca de los Servlets y JSP, por lo que se pasará a explicar cómo funciona el sistema de supervisión y sus respectivas clases Java.

Este sistema de supervisión está pensado para que sea gestionado por el administrador de la red, ya que será éste quien conozca las direcciones IP de las centralitas. Cuando éste abra la aplicación, deberá introducir las direcciones IP de las centralitas que desea monitorizar. No necesariamente debe introducir todas las direcciones que formen el clúster de Asterisk, si no que si lo desea, puede monitorizar una sola centralita, por ejemplo, ya que la aplicación está programada para que monitorice las centralitas que el usuario introduzca al inicio de la aplicación, de forma que no conectará automáticamente con todas las centralitas del clúster. En la siguiente imagen se muestra la portada de la aplicación:

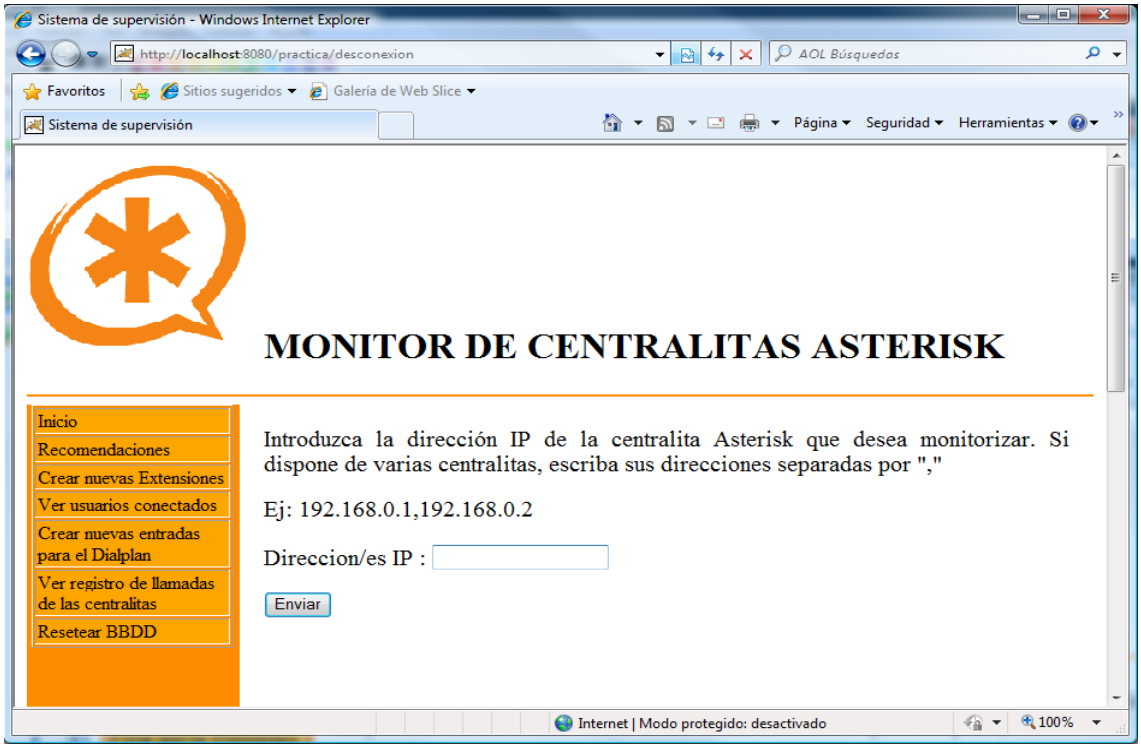

Figura 2: Portada del Sistema de Supervisión

Por tanto, una vez que se introduzcan las direcciones IP, al tratarse de un formulario, se enviará una petición POST al Servlet ServletObtencionIP, que pasamos a explicar a continuación.

4.2.6.2 ServletObtencionIP

Este Servlet es el encargado de recibir los datos del formulario que contendrán las direcciones IP que se desean monitorizar. Por tanto, dispondrá de un método doPost() formado como ya se explicó líneas más arriba. Con este Servlet se pretende conseguir lo siguiente:

- Comprobará que las direcciones IP introducidas por el administrador tengan un formato correcto.
- Si las direcciones son correctas, las almacenará en la sesión mediante un Vector, de forma que se puedan recuperar mientras la sesión esté activa para posteriores opciones.
- Comprobará que las bases de datos de esas centralitas estén activas o fuera de servicio.
- Por último, redirige la petición a la página principal de la aplicación para informar al administrador del estado de las bases de datos o de si las direcciones introducidas no tienen un formato correcto.

Una vez el administrador es redirigido a la página principal de la aplicación, podrá visualizar la información proporcionada por el Servlet anterior. Las siguientes imágenes muestran posibles salidas de la aplicación:

La primera de ellas, muestra la salida después de introducir una dirección IP correcta, en este caso 192.168.240.115, que no está disponible, y una dirección errónea, como sería a.b.c.d.

La segunda de ellas muestra la salida con dos direcciones correctas y disponibles, en este caso 192.168.240.115 (que ahora está activa) y 192.168.240.120. Si observamos la imagen de la topología de red en ambas, vemos que la primera sólo muestra una centralita Asterisk, puesto que sólo cuenta con una centralita para trabajar, mientras que en la segunda aparecen dos centralitas. Esto se debe a que la imagen varía según el número de centralitas que dispongamos para trabajar.

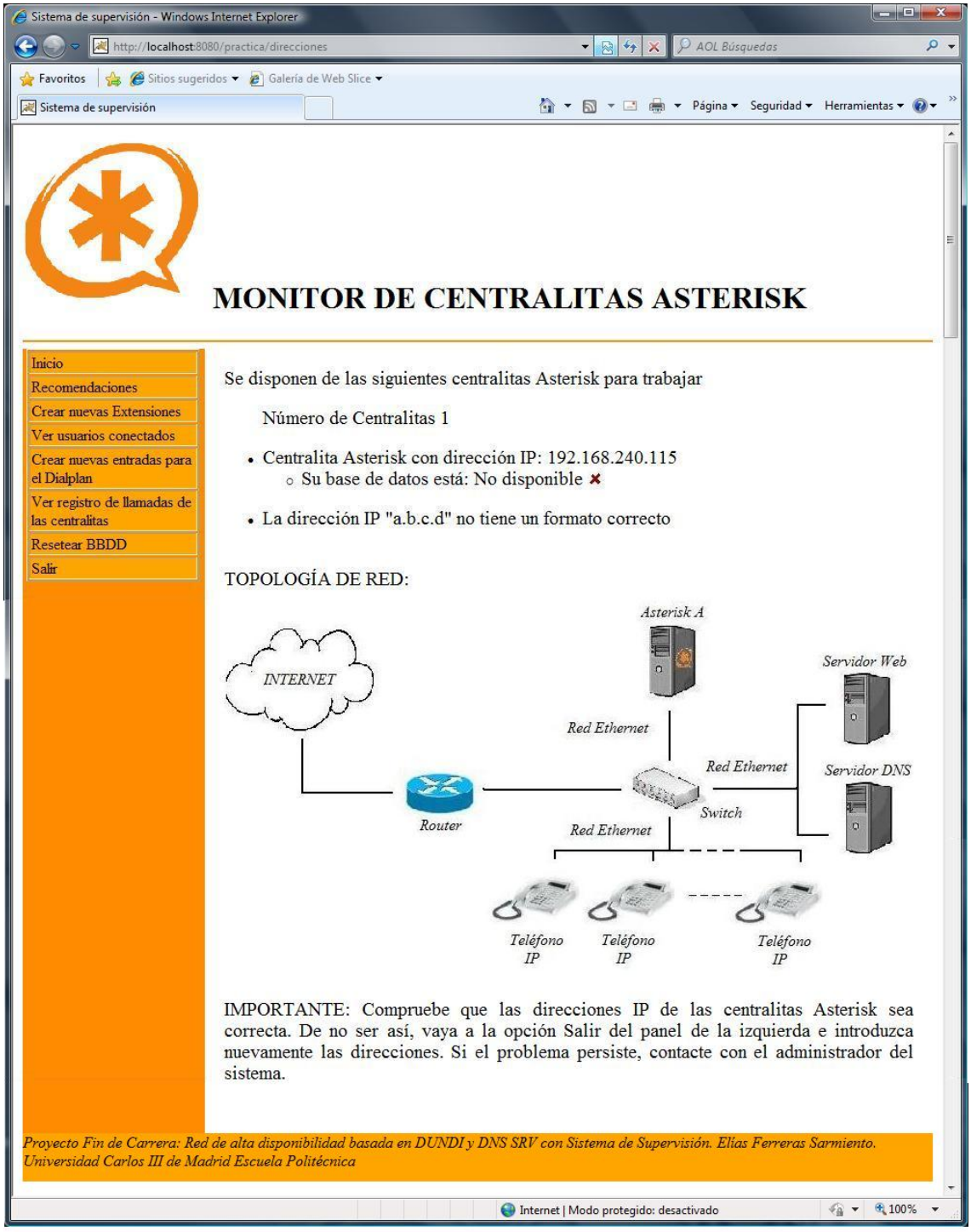

Figura 3: Página principal del Sistema de Supervisión

Figura 4: Página principal del Sistema de Supervisión

El administrador puede realizar diferentes acciones, que son:

- Inicio: Vuelve a la página principal y nuevamente comprueba el estado de las bases de datos.
- Recomendaciones: Muestra al usuario una serie de pautas para el correcto funcionamiento del sistema.
- Crear nuevas extensiones: Permite al administrador crear nuevas extensiones e insertarlas en la base de datos.

- Ver usuarios registrados: Muestra los usuarios registrados en las distintas centralitas.

- Crear entradas para el Dialplan: Como su propio nombre indica, permite crear nuevas entradas para el Dialplan, necesario cuando se crea una nueva extensión.

- Ver registro de llamadas: Muestra el registro de llamadas de las centralitas.

- Resetear BBDD: Introduce la dirección IP de la base de datos de la cual quiere resetear la tabla usuariosSIP.

- Salir: Elimina la sesión y vuelve a la portada de la aplicación.

Ahora que se conocen las diferentes opciones, se continúa explicando el funcionamiento de la aplicación.

Para que el administrador pueda hacer uso de las diferentes opciones, al menos debe haber introducido una dirección IP correcta. Por esta razón, cuando intente acceder a cualquiera de las opciones, la aplicación debe comprobar si en la sesión existe alguna dirección IP. Para ello hará uso del Servlet ServletCompruebaDireccion, al que enviará como parámetro en la URL la opción que el usuario desea. Por ejemplo, si el administrador desea crear nuevas extensiones, la URL tendría un aspecto similar a este: …./comprobar?opcion=crearExtensiones (comprobar es el nombre que se le asigna al Servlet en el fichero web.xml y opción el nombre que le damos para recuperar la variable en el Servlet). A continuación se muestra su funcionalidad.

4.2.6.3 ServletCompruebaDireccion

El siguiente Servlet, como ya sabemos, comprobará si existe en la sesión alguna dirección IP con la que poder trabajar. En esta ocasión, el Servlet implementará el método doGet() puesto que recibe los parámetros directamente de la URL. Por tanto, hará las siguientes acciones:

- Comprueba que existan direcciones en la sesión.
- Si las hay, redirige la petición a la opción requerida por el usuario (puede ser un Servlet o una página JSP), si no, reenvía la petición a la página principal y le notifica al usuario que no dispone de direcciones IP para trabajar.

En la siguiente imagen, se muestra qué ocurre cuando no hay direcciones correctas en la sesión. Se puede observar en la parte de la URL que se intentó ejecutar la creación de extensiones.

Figura 5: Página principal del Sistema de Supervisión

Ya se conoce el funcionamiento para comprobar si el administrador puede hacer uso de la aplicación, por lo que se continúa con las diferentes opciones del sistema.

Otra opción sería ver las recomendaciones. Aquí simplemente se redirige al administrador a una nueva página JSP donde visualiza una serie de pautas para el correcto funcionamiento de la aplicación. Esta opción es la única a la que se puede acceder sin disponer de direcciones en la sesión. La siguiente imagen muestra la página citada:

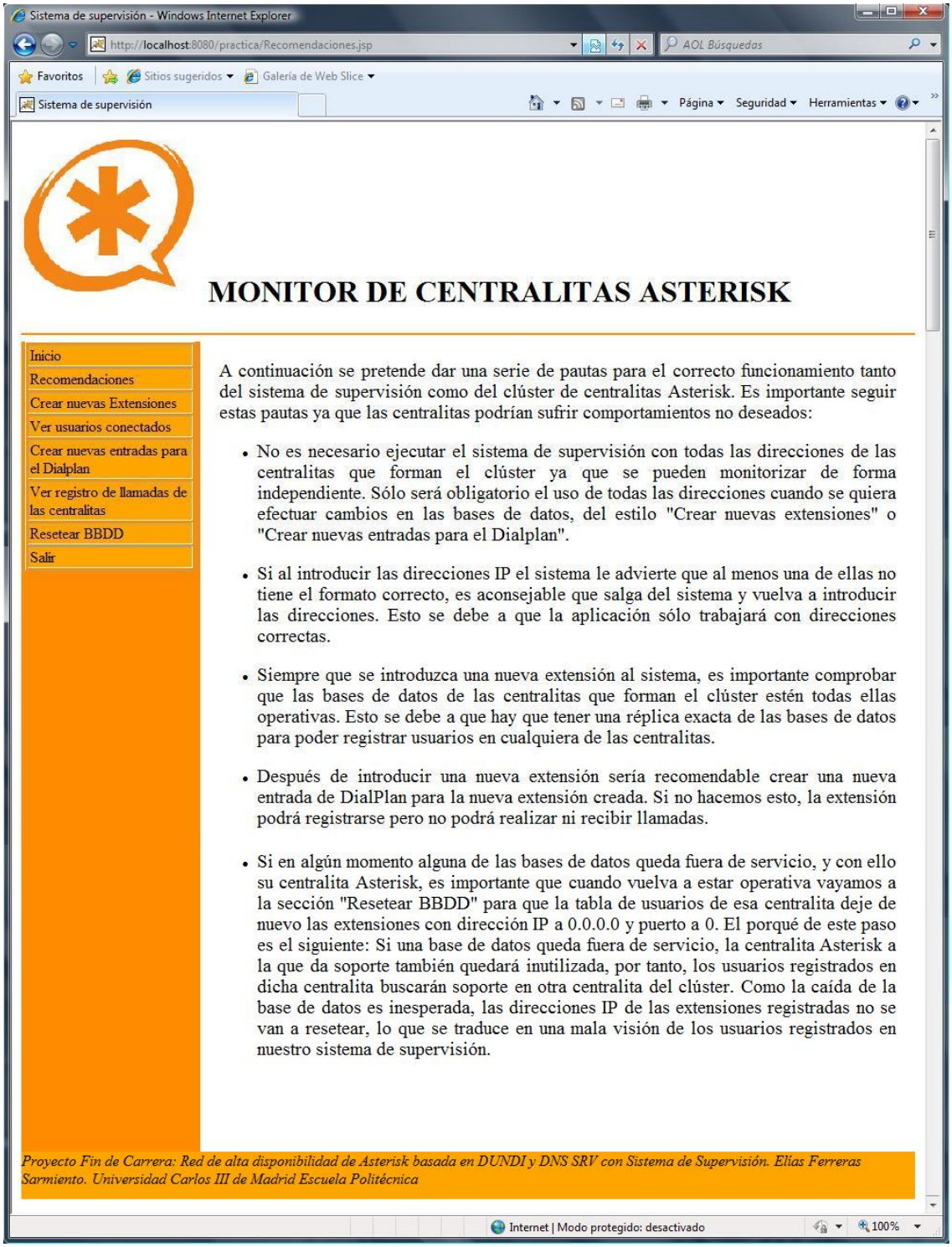

Figura 6: Recomendaciones del Sistema de Supervisión

La siguiente opción sería Crear nuevas extensiones, donde de igual modo, redirige al usuario a una nueva página JSP. En esta ocasión, deberá rellenar un formulario para introducir los datos en las bases de datos, que será el número de la extensión, el secret necesario y el username. Por tanto, estos datos se enviarán a un nuevo Servlet, ServletCrearExtensiones, que pasamos a explicar a continuación.

4.2.6.4 ServletCrearExtensiones

Este Servlet es el encargado de introducir nuevas extensiones en las bases de datos, en las direcciones que estén almacenadas en la sesión. Dispondrá del método doPost() puesto que los datos se envían de un formulario en petición HTTP POST. Por tanto, hará el siguiente cometido:

- Recupera los datos del formulario y las direcciones de la sesión.
- Comprueba que las bases de datos estén disponibles para realizar la inserción.
- Redirige al usuario a una página JSP donde se le notificará del estado de la inserción.

En las siguientes imágenes se muestra el formulario de inserción y el estado de dicha inserción en la base de datos:

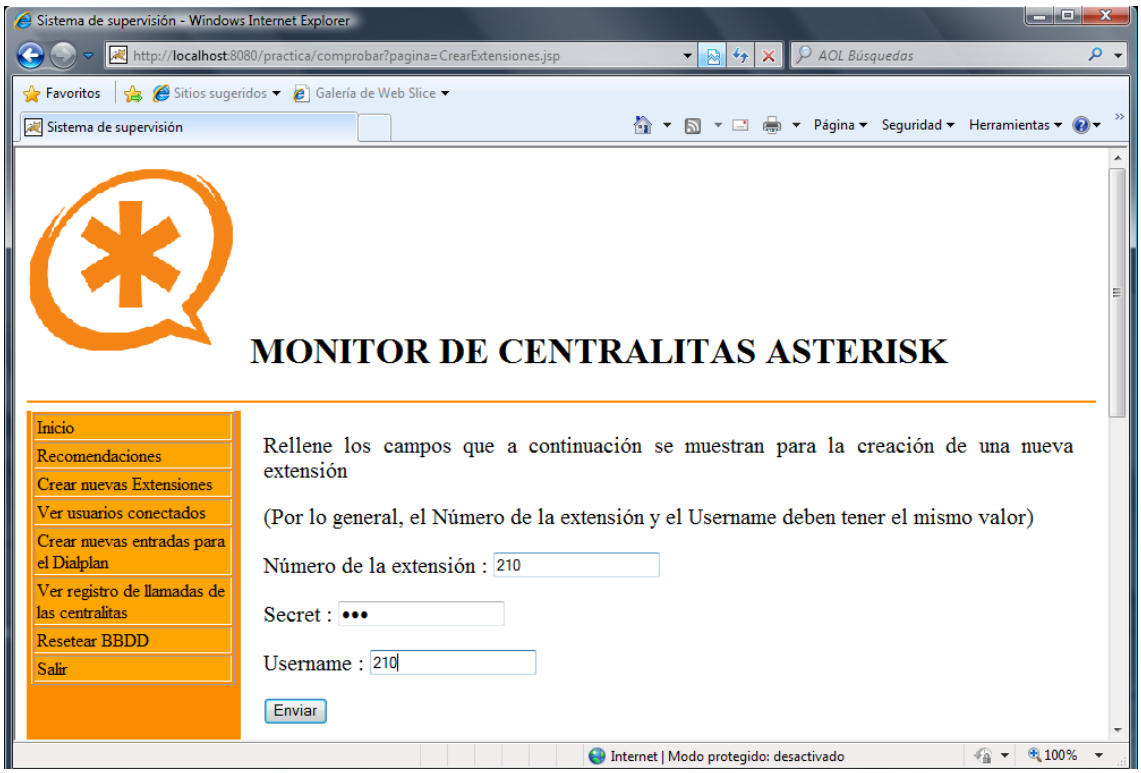

Figura 7: Crear nuevas extensiones en el Sistema de Supervisión

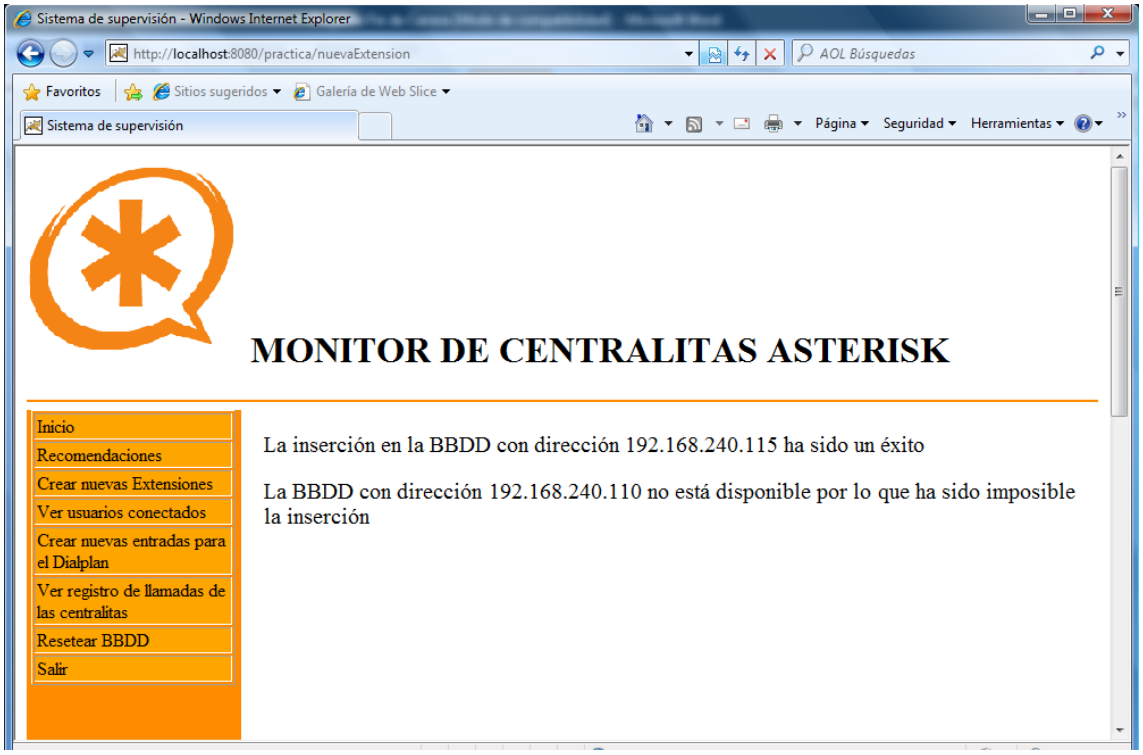

Figura 8: Resultado de crear nuevas extensiones

Continuando con las opciones, seguimos con Ver usuarios conectados. Como su propio nombre nos indica, esta opción nos permite ver qué usuarios están conectados en qué centralita. Para ello, redirige la petición al Servlet ServletVerUsuarios, que explicamos a continuación.

4.2.6.5 ServletVerUsuarios

Como se ha explicado, este Servlet nos va a permitir saber donde están registrados los usuarios. Debemos recordar que en la base de datos usuariosSIP que es la que utiliza la centralita para el registro de usuarios, distinguimos a los usuarios registrados del resto porque dispondrán de una dirección IP distinta de 0.0.0.0. Para dicho Servlet, se hará uso del método doGet(), puesto que es una petición "normal" de HTTP. Tendrá el siguiente cometido:

- Recupera las direcciones IP de la sesión para consultar las bases de datos.
- Como siempre, comprueba que las bases de datos estén disponibles.
- Recupera los datos necesarios para mostrarlos al administrador.
- Por último, redirige al usuario a una nueva página JSP donde se le muestra el resultado de la consulta, ya sea para mostrarle los usuarios registrados o para notificarle que la consulta no se pudo realizar por diferentes motivos, que puede ser porque la base de datos no esté disponible o porque haya un error en la consulta a la base de datos.

A continuación se muestra la imagen con la consulta de usuarios de dos centralitas, una de ellas fuera de servicio:

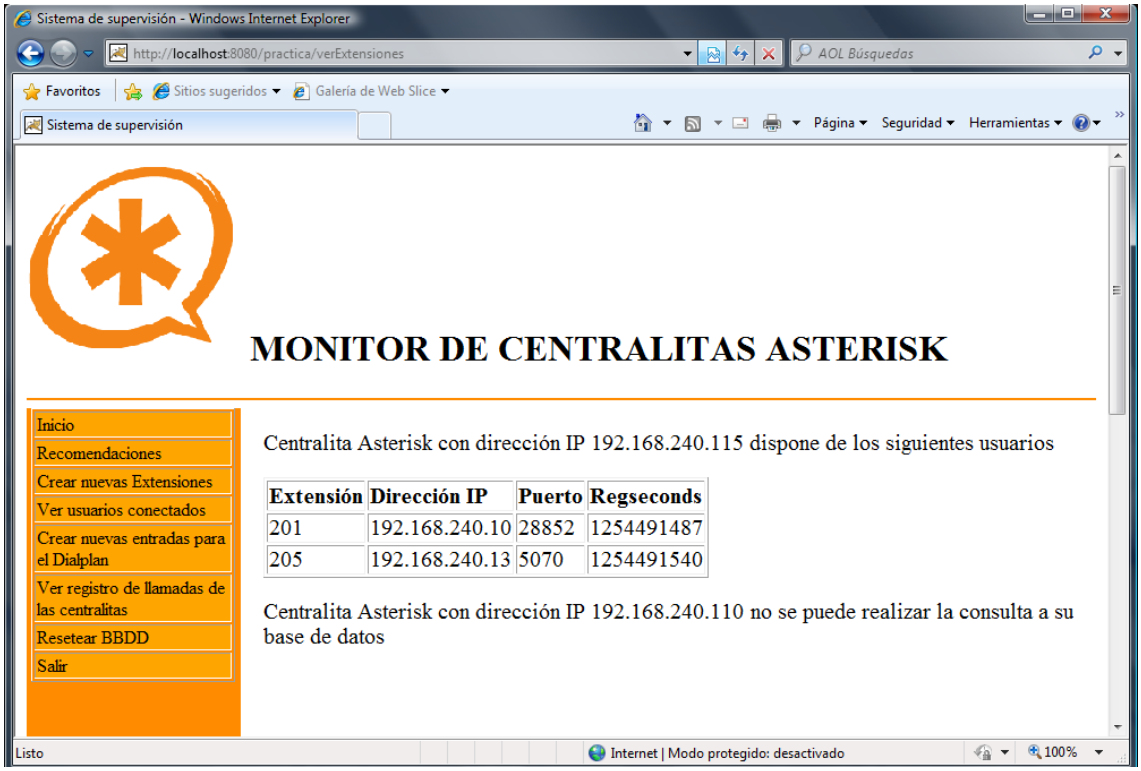

Figura 9: Ver usuarios conectados en el Sistema de Supervisión

La siguiente opción es Crear nuevas entradas para el Dialplan. Esta opción es exactamente igual que la de crear nuevas extensiones, con la diferencia de que los datos a insertar van a una tabla diferente y que los datos son diferentes. Por tanto, redirige al usuario a una nueva página JSP donde dispondrá de un nuevo formulario con las opciones necesarias para la inserción de una nueva entrada en el Dialplan, que en este caso serán el contexto al que hace referencia la llamada, el número de la extensión a la que se quiere aplicar, la prioridad de la sentencia, el tipo de aplicación y los datos para ejecutar dicha aplicación. Si recordamos el módulo de las bases de datos, corresponde con las cinco entradas de la columna Dialplan. Los datos de este formulario se enviarán al ServletCrearEntradasDP, que se explica a continuación.

4.2.6.6 ServletNuevaDP

Este Servlet dispondrá de un método doPost() por la misma razón que en el Servlet ServletCrearUsuarios. En esta ocasión, creará las entradas para el Dialplan necesarias para que una nueva extensión pueda realizar llamadas. Su estructura de ejecución es la siguiente:

- Recupera los datos del formulario y las direcciones de la sesión.
- Comprueba que las bases de datos estén disponibles para realizar la inserción.
- Redirige al usuario a una página JSP donde se le notificará del estado de la inserción.

En las siguientes imágenes se muestra el formulario y la página de estado de la inserción:

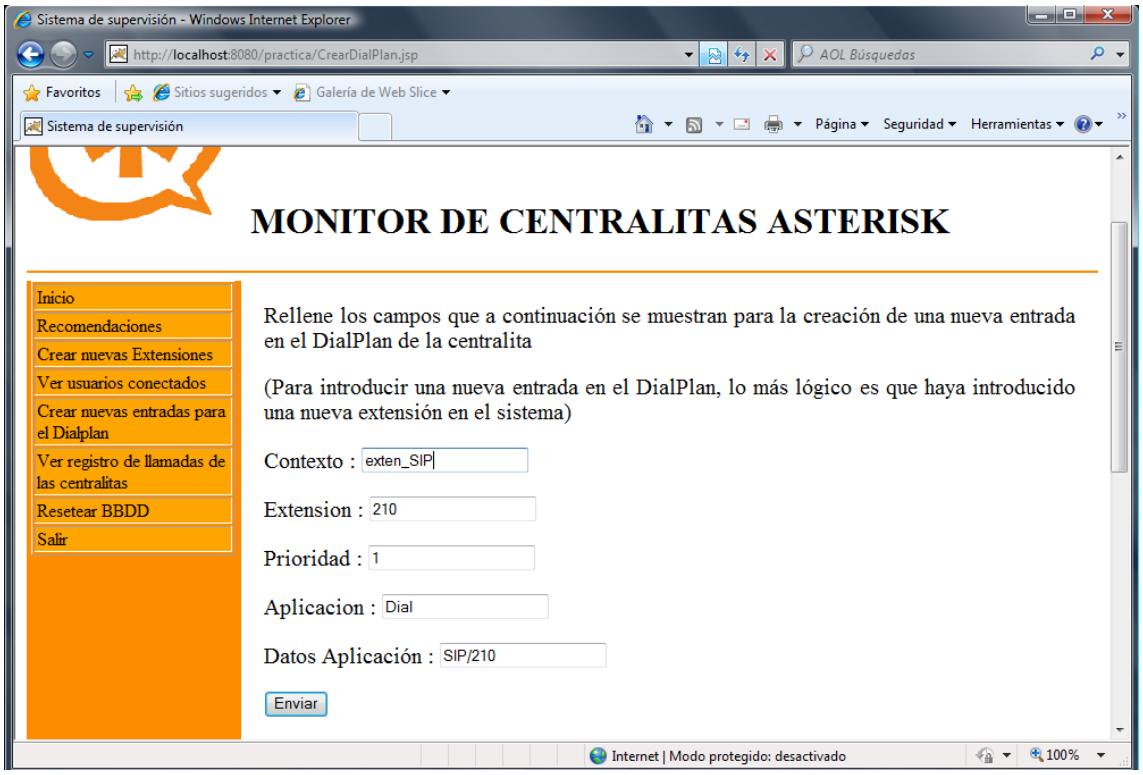

Figura 10: Crear nueva entrada del dialplan en Sistema de Supervisión

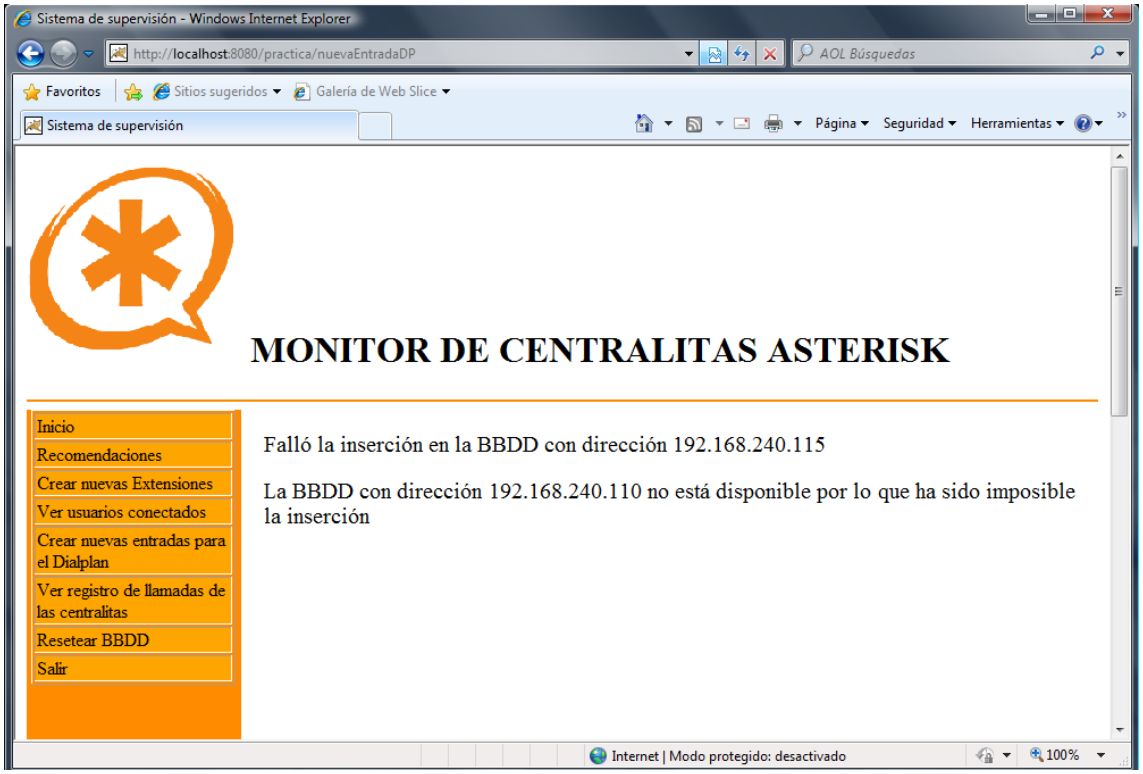

Figura 11: Resultado de la creación de una entrada de dialplan

La siguiente opción corresponde con poder ver el registro de llamadas de las distintas centralitas. La aplicación enviará el control al Servlet ServletVerCDR para dicha función, por tanto, tendrá la siguiente forma.

4.2.6.7 ServletVerCDR

Este Servlet es el encargado de obtener los datos de la tabla CDR para poder visualizar el registro de llamadas de las centralitas. En esta ocasión el método a implementar es doGet(). Las acciones a realizar son las siguientes:

- Recupera las direcciones IP de la sesión para consultar las bases de datos.
- Comprueba que las bases de datos estén disponibles.
- Recupera los datos necesarios para mostrarlos al administrador.
- Por último, redirige al usuario a una nueva página JSP donde se le muestra el resultado de la consulta, ya sea para mostrarle una tabla con el registro de las llamadas o para notificarle que la consulta no se pudo realizar por diferentes motivos, que puede ser porque la base de datos no esté disponible o porque haya un error en la consulta a la base de datos.

La siguiente imagen muestra la consulta a dos centralitas, una de ellas disponible y la otra fuera de servicio:

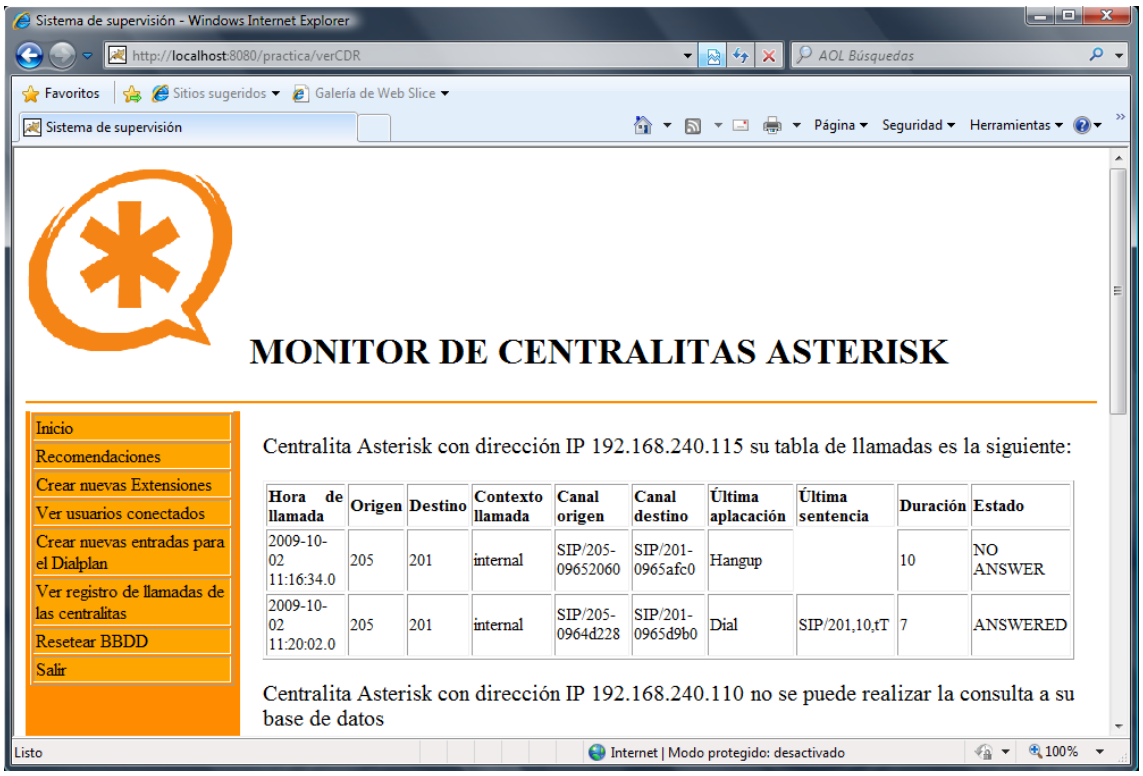

Figura 12: Ver registro de llamadas en el Sistema de supervisión

Continuando con las opciones, la siguiente corresponde al reseteo de las bases de datos de su tabla usuariosSIP. Esto no quiere decir que se eliminen las filas de la tabla, si no que lo que se pretende es dejar las direcciones IP y los puertos de las extensiones

con valores 0.0.0.0 y 0 respectivamente. El porqué de esta opción está explicado en el apartado de recomendaciones. En esta ocasión no se va a trabajar con las direcciones de la sesión por motivos de seguridad, por lo que se volverá a introducir las direcciones IP de la base de datos que se desee resetear. Por tanto, el administrador será redirigido a una nueva página JSP donde dispondrá de un formulario para introducir la dirección IP de la base de datos. Posteriormente, el control pasará a un nuevo Servlet, ServletReset, para llevar a cabo las opciones pertinentes. A continuación se explica detalladamente.

4.2.6.8 ServletReset

Este Servlet es el encargado de resetear la tabla usuariosSIP de las bases de datos introducidas por el administrador. Como nuevamente recibimos los datos mediante un formulario, se implementará el método doPost(). Las acciones a seguir son las siguientes:

- Obtiene las direcciones IP del formulario y comprueba que sean correctos en cuanto a su formato.
- Comprueba que la base de datos de la dirección esté disponible. Siendo así resetea la tabla usuariosSIP.
- Redirige al usuario a una página JSP para indicarle el estado del reseteo, es decir, si se ha podido realizar con éxito o si la base de datos no estaba disponible.

Las siguientes imágenes muestran el formulario para introducir la dirección IP y el estado del reseteo:

Figura 13: Página de reset en el Sistema de Supervisión

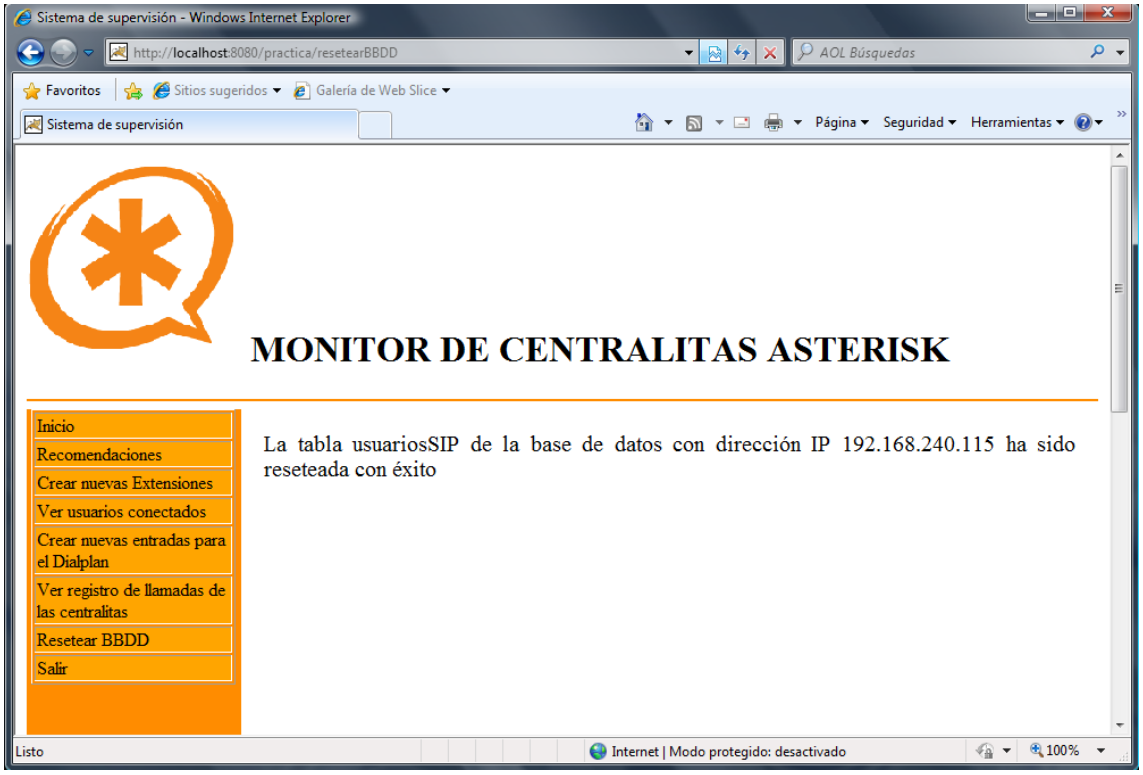

Figura 14: Resultado del reset

Por último, el administrador tiene la opción de salir del sistema, pero pasará el control a un último Servlet, ServletDesconexion cuya finalidad es muy sencilla pero útil.

4.2.6.9 ServletDesconexion

Este último Servlet es el encargado de finalizar la sesión, de forma que las direcciones IP almacenadas desaparezcan y redirija al administrador a la portada de la aplicación. Aparentemente puede resultar poco útil este Servlet ya que cada vez que se introduzcan nuevas direcciones se crea una sesión para almacenarlas, pero por seguridad y evitar comportamientos no deseados, conviene finalizar la sesión previamente.

Para terminar, hay que mencionar que las funciones que realizan los Servlets, como conectar con la base de datos, comprobar si está disponible, consultar datos, insertarlos, etc, se realizan gracias a que se ha implementado una clase Java (no es Servlet) donde se han implementado todos estos métodos. Los Servlets crean un Objeto de esta clase y utilizan sus métodos. En el ANEXO G se va a mostrar el documento generado por JAVADOC donde se explican todas las clases y donde se puede consultar esta última clase mencionada, Conexión, para ver todos los métodos y funciones de ésta. Por último, en el ANEXO H se muestra el código fuente de las distintas páginas JSP creadas para la aplicación.

Capítulo 5

Material de trabajo y Pruebas

En el siguiente apartado se dará a conocer todos los elementos utilizados para la realización del proyecto.

En primer lugar, necesitamos varias máquinas para poder simular la red. Como se pretende crear un clúster de Asterisk, de momento basta con utilizar dos equipos para las centralitas, puesto que la red a implementar será una red de pruebas. Por otro lado, se va a necesitar dos máquinas para los servidores DNS y otra para el servidor web donde irá alojado nuestro sistema de supervisión para las centralitas. Por último, serán necesarios los terminales para realizar las pruebas de llamadas.

Una vez conocemos el número de máquinas necesarias, se puede observar que es un número elevado de equipos y no es fácil disponer de tal cantidad, por lo que se decide hacer uso del software VMWare Server versión 1.07. Este programa nos permite crear tantas máquinas virtuales como nos permita el equipo donde se encuentra instalado. Este programa se instalará en un equipo PC sobremesa K7 con procesador AMD a 3,2 GHz y 1 GB de memoria RAM. El sistema operativo que correrá en dicha máquina será Windows XP Home Edition.

Como ya se ha comentado, necesitaremos dos máquinas para las centralitas, dos más para los servidores DNS y una más para el servidor web, aunque este último irá alojado en una máquina física, por lo que será necesaria la creación de cuatro máquinas virtuales. Las características físicas de estas máquinas pueden variar debido a que dependen, en parte, al hardware disponible en el equipo físico donde se encuentran instaladas, por lo que por ejemplo, la memoria RAM, puede variar de unas a otras. Lo que no variará será el sistema operativo, instalando en todas ellas Debian 5.0 conocido con el nombre de Lenny, siendo actualmente la última versión estable. Es muy importante comentar en este punto que debemos instalar en todas las máquinas virtuales openssh-server, puesto que como veremos más adelante, la configuración de éstas se hará de forma remota vía SSH.

Las centralitas, como cabe de esperar, usarán el software Asterisk, en este caso la versión 1.6.1.1 y asterisk-addons-1.6.1.1 para el uso de bases de datos MySQL. Para consultar su instalación, puede consultarse el ANEXO D. Los servidores DNS utilizarán el software BIND 9.0, como ya se comentó en apartados anteriores. Por último, el servidor web, al que se le instalará el servidor Tomcat, como ya se comentó anteriormente.

Se dispone de un ordenador portátil, en esta ocasión AMD Turion X2 64 de doble núcleo a 2,2 GHz y 4GB de memoria RAM, se decide trabajar en remoto con las máquinas virtuales para no saturar de carga computacional al ordenador donde residen dichas máquinas y no ralentizar su trabajo. Para poder trabajar en remoto desde el ordenador portátil al equipo K7, se instalará en el primero de éstos el programa putty, siendo éste un cliente Telnet y SSH para Windows que posee multitud de funciones e

interesantes opciones. Para la realización del trabajo, la conexión se hará vía SSH, de forma que sea segura y encriptada, de ahí que en las máquinas virtuales se instalase openssh-server. También cabe mencionar que el servidor web Tomcat se instalará en esta misma máquina.

Por último, queda hablar de los terminales. Como se sabe, debían tener una característica concreta, y era tener soporte para DNS SRV. Por desgracia, no todos los teléfonos disponen de tal servicio. Para la realización del proyecto, se han usado tres tipos diferentes de softphone, X-lite 3.0, Zoiper y Twinkle, siendo este último el único con soporte DNS SRV. En el apartado de pruebas, se explicará con más detalle el por qué de utilizar estos tres softphone y no usar solamente Twinkle, que en un principio parece el más indicado puesto que cubre con el requisito fundamental.

Después de explicar el material de trabajo que se va a utilizar para la realización del proyecto, se puede exponer como se ha llevado a cabo el trabajo dividido en diferentes fases debido al gran número de módulos que intervienen sólo en la simulación de la red, por lo que se ha decidido implementarla poco a poco y asegurarnos del correcto funcionamiento de cada uno de ellos.

En primer lugar, se configurará el servidor DNS para comprobar que los teléfonos son capaces de registrarse vía DNS SRV. Como se ha podido leer unas líneas más arriba, sólo el softphone Twinkle dispone de servicio DNS SRV, por lo que para esta primera prueba será necesario su uso. El problema de este softphone es que sólo está disponible para sistemas Linux. Esto implica que una de las máquinas virtuales creadas deberá tener entorno gráfico para poder hacer uso del softphone. Recordar que como las máquinas virtuales harán función sólo y exclusivamente de servidor, se instalarán sin entorno gráfico, por lo que se ejecutarán por línea de comandos. Twinkle está disponible en paquetes Linux, por lo que se puede instalar con apt-get. Una vez se tenga el softphone instalado y el servidor DNS configurado tal y como se explicó en el módulo correspondiente, podemos configurar el softphone de la siguiente manera:

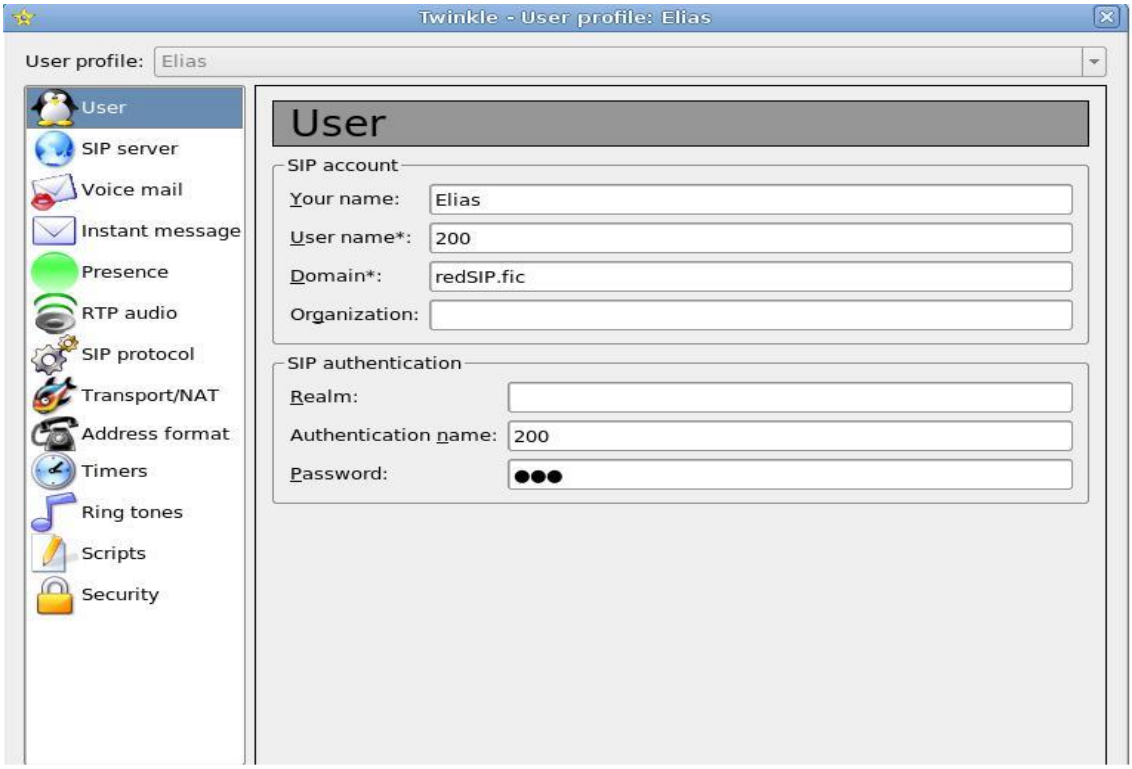

Figura 15: Configuración softphone Twinkle

Como se puede apreciar, se está configurando la extensión 200 y se está especificando en el campo Domain el nombre de dominio de nuestra red. No hay que hacer nada respecto a DNS SRV puesto que lo hace el softphone de manera interna una vez puesto el nombre de dominio en el campo anteriormente nombrado.

Cabe destacar que en este punto ya están configurados los usuarios en RealTime, por lo que aparte de ver si el servidor DNS hace su función de registro de forma correcta, también vamos a observar si la centralita es capaz de acceder a la base de datos y obtener la información pertinente de la tabla de usuarios. Con todo esto, ya sólo falta comprobar que el teléfono se registra con la extensión configurada. Para ello, podemos acceder a la consola de asterisk, conocida como Asterisk CLI, donde podemos debugear y comprobar el funcionamiento de asterisk. Para acceder a ella debemos ejecutar ***asterisk –r***, pero si queremos ajustar el nivel de debug y verbose de forma que obtengamos mayor información acerca de mensajes de error, warning, etc, se puede usar el comando ***asterisk –rvvvvvvvvvvv*** siendo el número de 'v' el que ajusta el nivel de debug y verbose (a mayor número de 'v' más información). Una vez dentro de la consola CLI podemos registrar el teléfono y ver el mensaje que nos aparece, en nuestro caso será el siguiente:

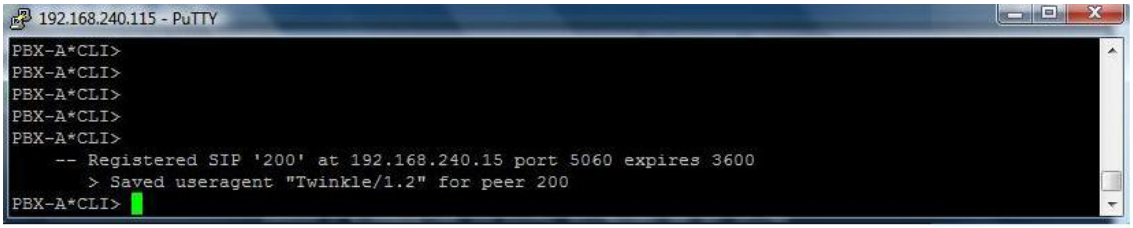

Figura 16: Registro de extensión 200

Se puede observar que el registro ha sido un éxito puesto que nos encontramos con la extensión 200 registrada con la dirección IP y puerto de la máquina donde se encuentra instalado Twinkle. Si se observa también el softphone, aparece en pantalla que el terminal se ha registrado con éxito:

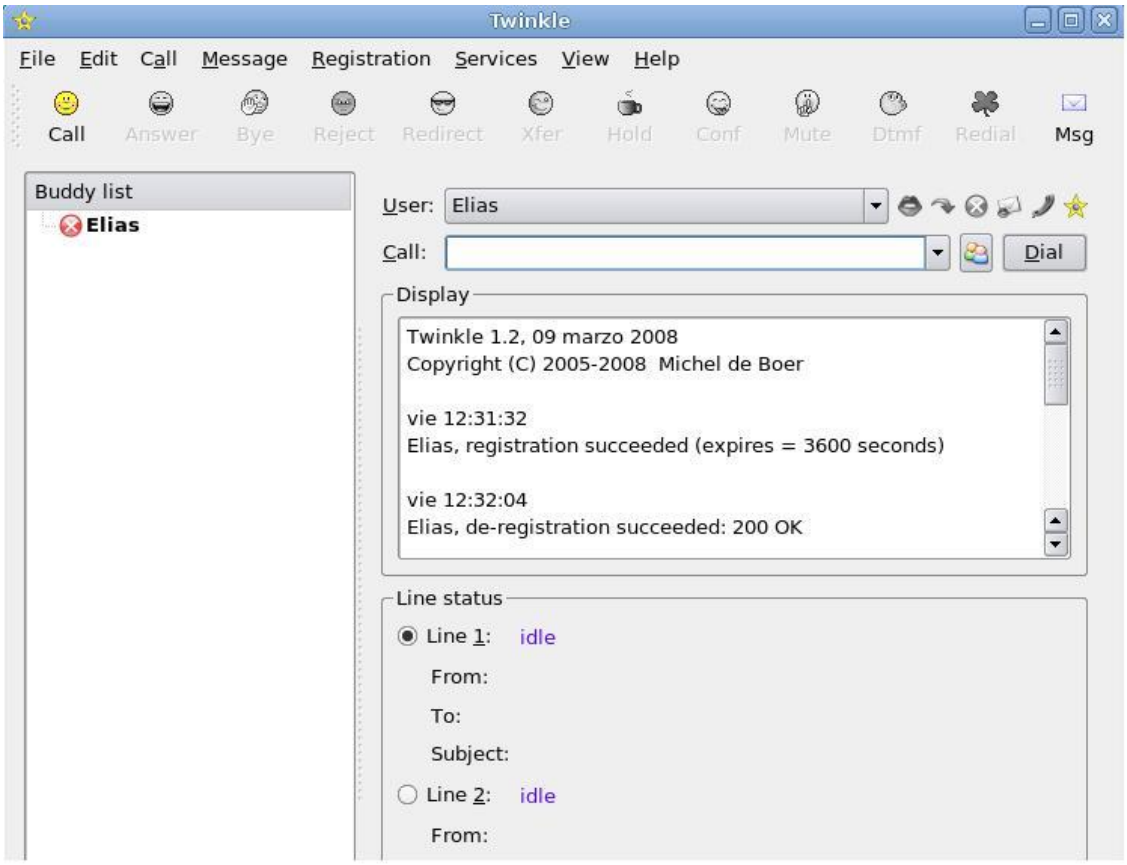

Figura 17: Registro de extensión en Twinkle

Como se está probando el uso de RealTime con el registro de usuarios, es bueno acceder a la base de datos y comprobar que efectivamente en los campos de dirección IP y puerto han sido modificados con los valores del nuevo terminal

Antes del registro:

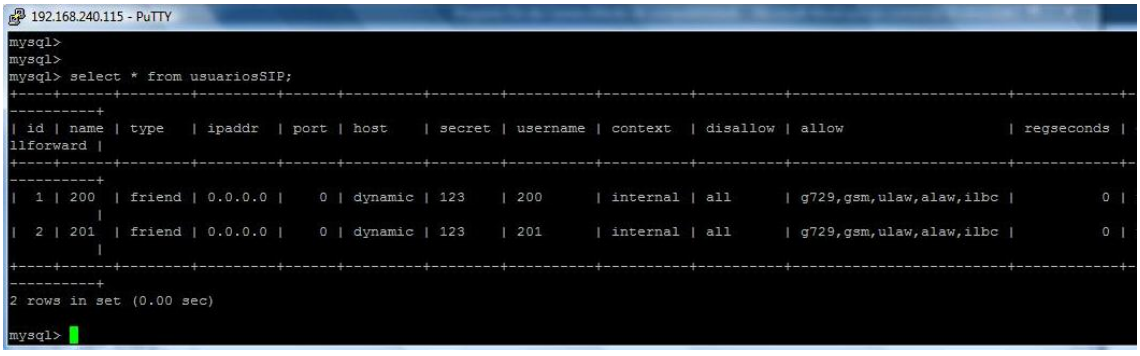

Figura 18: Base de datos antes del registro

Después del registro:

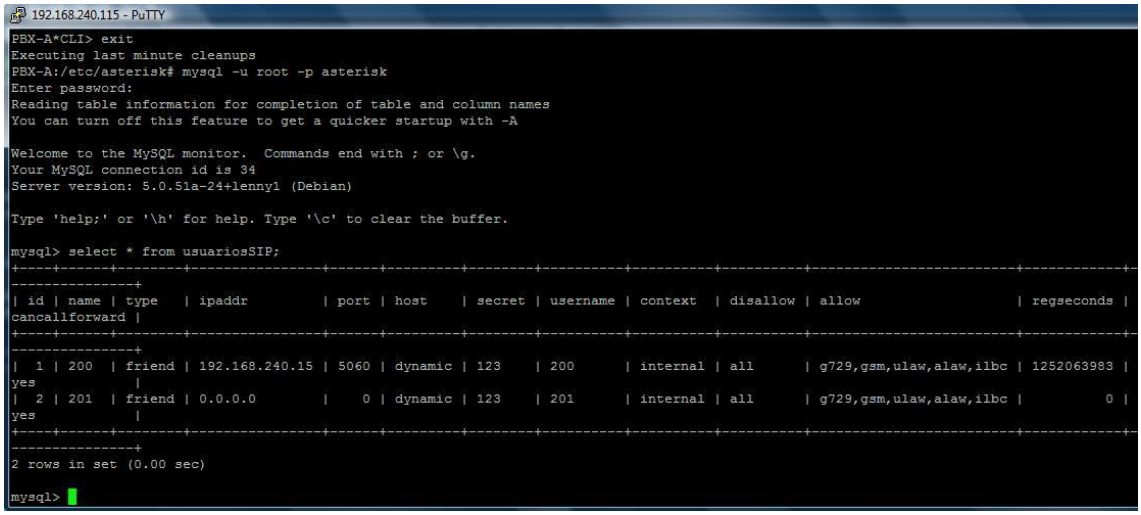

Figura 19: Base de datos después del registro

(en estas dos últimas imágenes faltan campos de la tabla usuariosSIP que no aparecen debido a la dimensión de ésta. Aun así, no son campos relevantes para comprobar si se ha efectuado el cambio en la tabla)

Como cabía de esperar, se puede apreciar como el campo *ipaddr* y *port* han sido modificados con la información del softphone Twinkle. Además, se observa también como cambia el valor del campo regseconds.

Por último, se ha decidido realizar una captura con el programa Wireshark (antiguo Ethereal) para visualizar el intercambio de tramas. Este programa es un analizador de protocolos que nos permite visualizar las tramas que circulan por nuestra red. De esta forma, se puede observar el intercambio de mensajes entre terminal y DNS, y viceversa, y entre terminal y centralita, y viceversa.

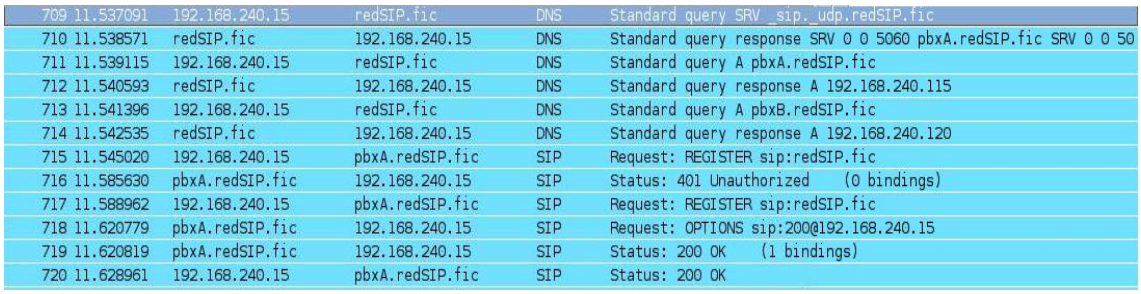

Figura 20: Captura de tramas Wireshark

(por problema en el tamaño de la imagen, sólo se muestran las tramas capturadas)

Como se aprecia, la dirección IP 192.168.240.15, que corresponde con la máquina donde se encuentra instalado Twinkle, realiza una petición SRV al dominio redSIP.fic, correspondiente al servidor DNS. Éste le envía la información pertinente de las dos centralitas, por lo que el siguiente paso será pedir la dirección IP de éstas por parte del terminal. Una vez tiene las direcciones IP, basta con efectuar la petición de

registro a una de ellas, en este caso a pbxA, con dirección IP 192.168.240.115. Se observa como el registro termina con la confirmación por ambas partes de un mensaje de OK.

Hecho todo esto, podemos concluir que nuestro servidor DNS está listo para recibir peticiones de registro y que la centralita es capaz de acceder a la base de datos para consultar y modificar la información de los usuarios registrados. Para terminar, hay que comprobar que si la centralita en la cual el terminal está registrado se cae, es capaz de enviar una nueva petición de registro a la otra centralita operativa. Lo más fácil es poner un tiempo de registro bajo para obligar a que pasado ese tiempo tenga que volver a registrarse y comprobar que lo hace en la centralita operativa. Con todo esto, podemos pasar al siguiente punto, conseguir que usuarios de distintas centralitas sean capaces de llamarse entre sí.

En esta prueba no será necesario hacer uso de nuestro servidor DNS, ya que lo que nos interesa es conseguir que los usuarios sean capaces de llamarse cuando se encuentran en centralitas distintas. Es en este punto por lo que se decide utilizar otro tipo de softphone, en concreto X-lite 3.0 y Zoiper. Esto se debe fundamentalmente a que estos dos softphones permiten instalarse en sistemas Windows, lo que nos da mayor flexibilidad a la hora de trabajar con las máquinas. Esto supone instalar los programas (los softphones) en el portátil y realizar las llamadas desde éste, sin necesidad de sobrecargar el equipo donde se encuentran las máquinas virtuales, ya que era ahí también donde se encontraba instalado Twinkle. Aclarado el por qué del cambio de softphone, se puede pasar a explicar la prueba llevada a cabo.

Es fácil intuir que los módulos que vamos a poner en práctica para esta prueba son el protocolo DUNDi y RealTime para la ejecución de llamadas, más concretamente para el dialplan. Por tanto, debemos configurar DUNDi como ya se explicó en su correspondiente módulo. Ahora, para simular que el servidor DNS ha servido direcciones IP de centralitas distintas a dos de los terminales, basta con registrar los teléfonos directamente en las dos centralitas que disponemos para crear el clúster. En las siguientes imágenes se muestra cómo hacerlo.

Para el terminal X-Lite accedemos a la configuración de SIP y escribimos lo que muestra en la imagen:

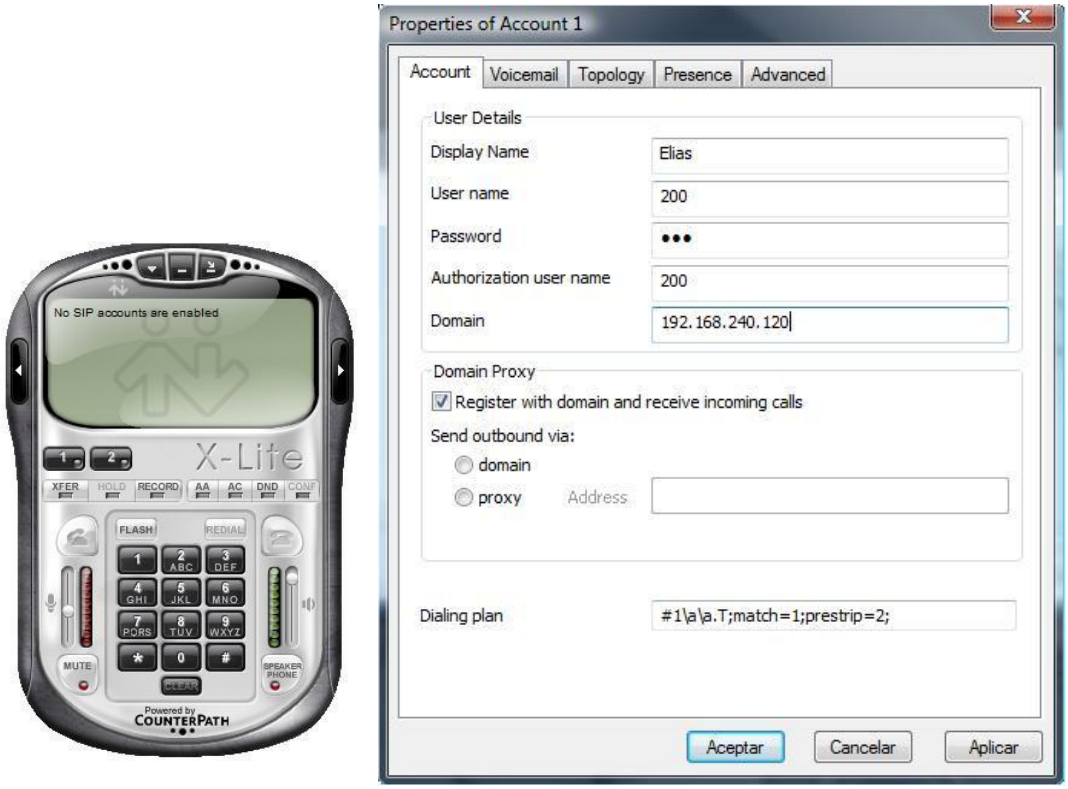

Figura 21: Configuración softphone X-lite

Como se puede observar, se ha configurado en este terminal la extensión 200 en la centralita con dirección IP 192.168.240.120 (PBX-B). Una de las diferencias respecto a la configuración en el terminal de Twinkle es que en el campo Domain se pone directamente la dirección IP de la centralita, ya que no vamos hacer uso de DNS SRV en esta prueba.

Para el terminal de Zoiper, se realiza de la siguiente forma:

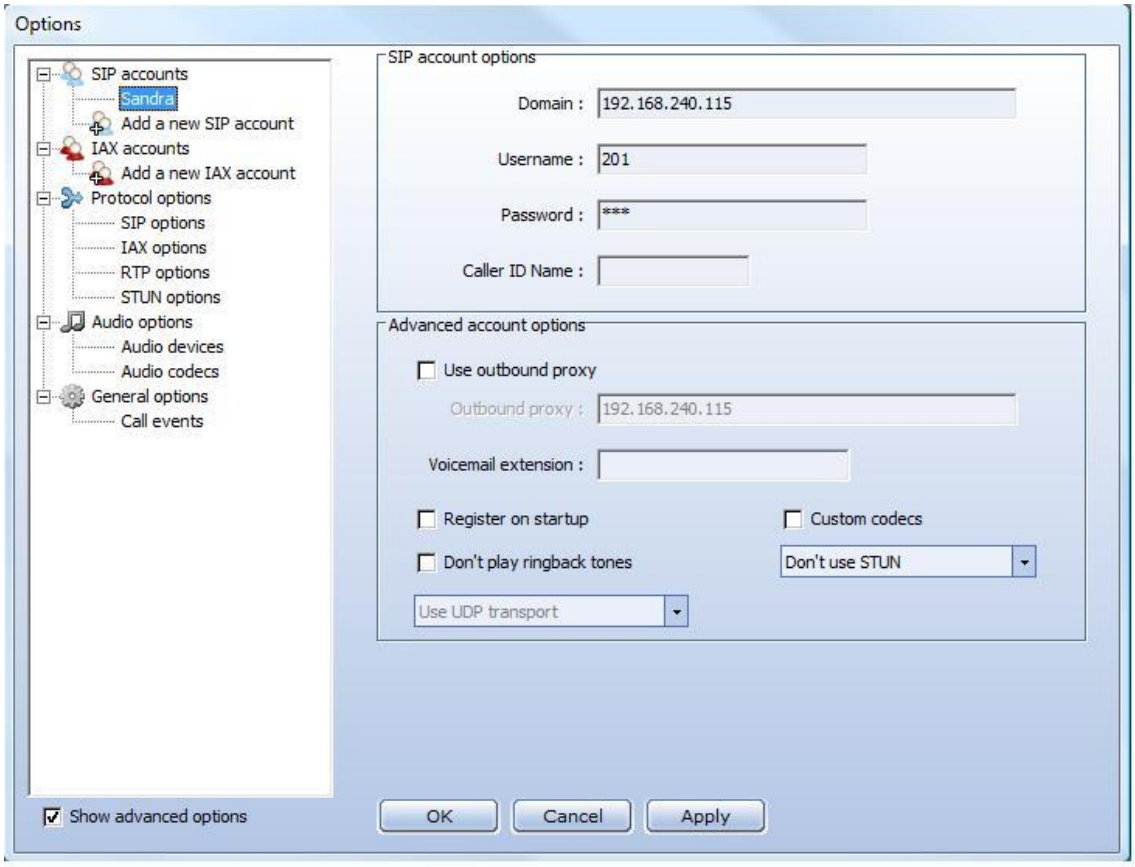

Figura 22: Configuración softphone Zoiper

En este terminal, se ha configurado la extensión 201 en la centralita con dirección IP 192.168.240.115 (PBX-A).

Una vez tengamos los teléfonos registrados, antes de realizar la llamada, vamos a comprobar en la consola de los dos Asterisk si el protocolo DUNDi está bien configurado y nos aporta información de la centralita contraria. Nuevamente, accedemos a la consola de Asterisk con el comando ***asterisk –rvvvvv***. Una vez dentro, ejecutamos el siguiente comando en ambas centralitas ***dundi show peers***, obteniendo la siguiente información:

Ejecución del comando en la centralita PBX-A (dirección IP 192.168.240.115; dirección MAC 00:0c:29:a4:1e:30)

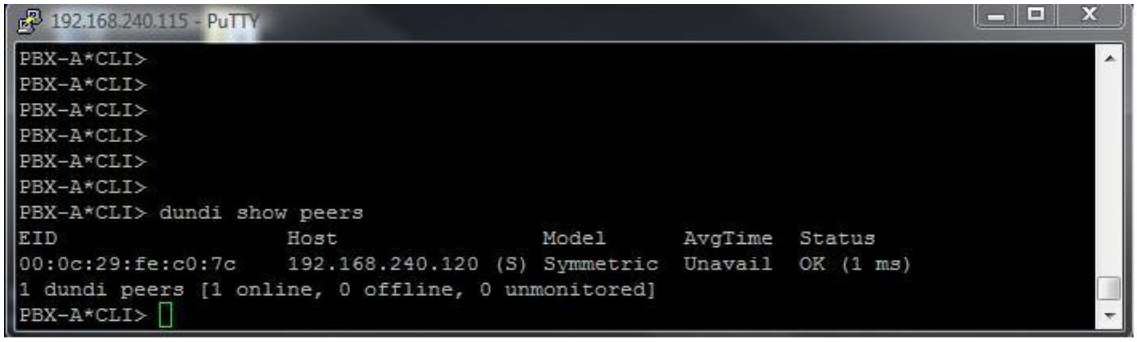

Figura 23: Ejecución del comando *dundi show peers* PBX-A

Se observa que hay un peer creado, que corresponde a la centralita PBX-B, representado por su entityid (dirección MAC) y su dirección IP. Además observamos que el status es OK, lo que hace intuir que la ejecución de este comando en la centralita restante dará la misma salida pero con la información de esta.

Ejecución del comando en la centralita PBX-B (dirección IP 192.168.240.120; dirección MAC 00:0c:29:fe:c0:7c)

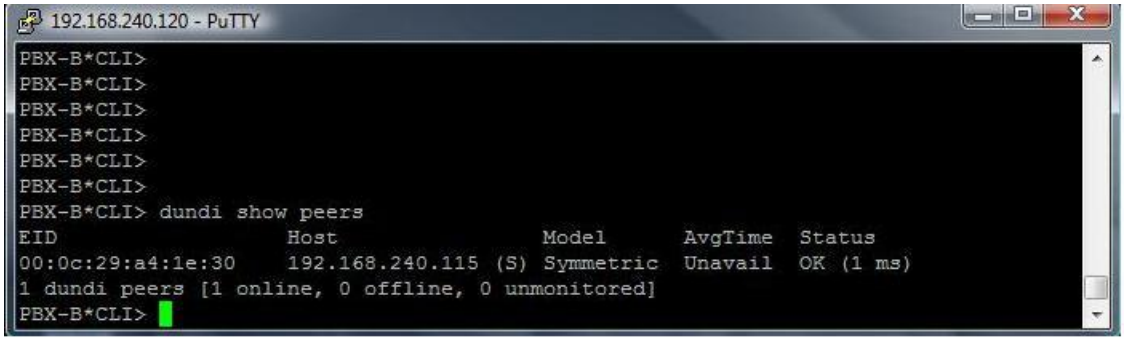

Figura 24: Ejecución del comando *dundi show peers* PBX-B

Al igual que en el otro Asterisk, cabía de esperar que también aquí se encuentre el peer correspondiente a la centralita PBX-A.

De momento se observa que el canal entre las dos centralitas se ha creado, puesto que nos aparece el peer de la otra centralita. Ahora se puede comprobar si nos aporta información del otro peer, ejecutando el siguiente comando, ***dundi query direcciónMAC@troncalIAX*** y obteniendo la siguiente salida:

Ejecución del comando en la centralita PBX-A (la dirección MAC corresponde al entityid de la centralita de la cual queremos obtener información)

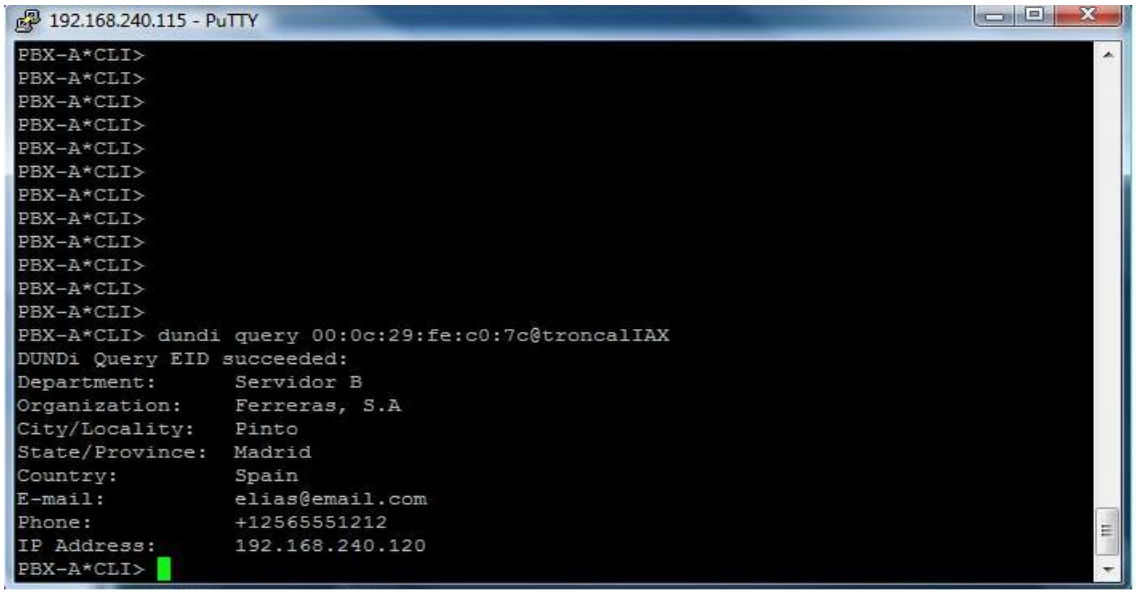

Figura 25: Ejecución del comando *dundi query* PBX-A

Como se observa, nos aporta información de la centralita PBX-B, perteneciente al archivo de configuración dundi.conf del apartado [general] (esto se puede ver en el ANEXO en el archivo de configuración dundi.conf).

Ejecución del comando en la centralita PBX-B

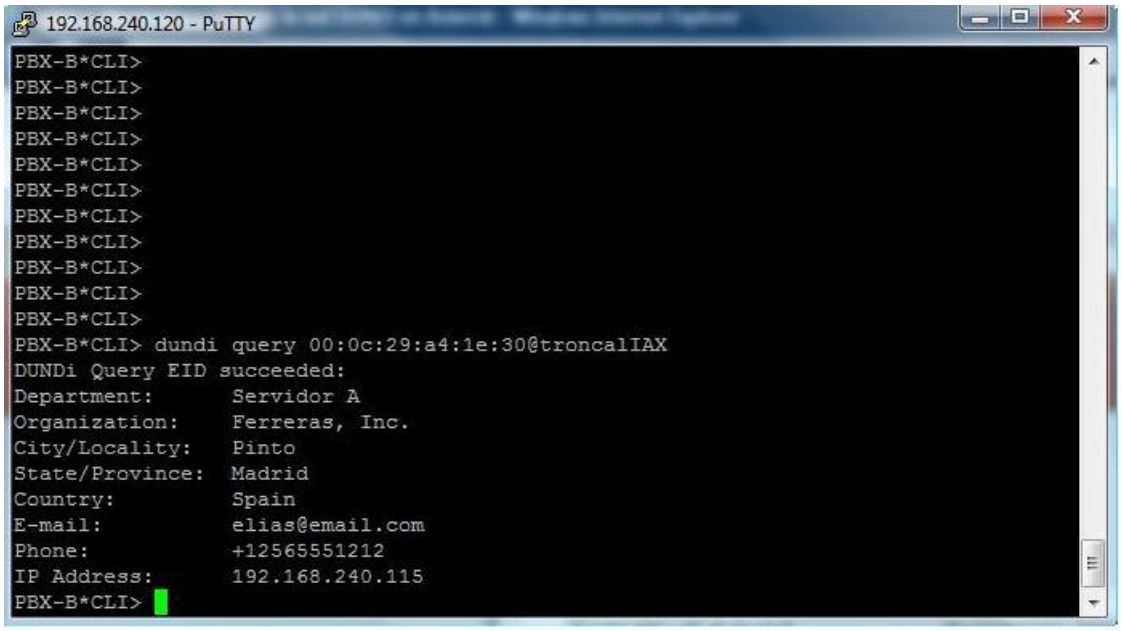

Figura 26: Ejecución del comando *dundi query* PBX-B

Nuevamente, vemos como nos aporta la información de la centralita PBX-A.

La salida del comando nos ha aportado información del otro peer, por lo que de momento nuestra configuración es correcta. Ya sólo queda comprobar si son capaces de distribuir la información de las extensiones que tienen registradas, vital para poder realizar las llamadas entre ellas. Para esto último, ejecutamos el siguiente comando, con la extensión registrada en la centralita contraria, ***dundi lookup extension@troncalIAX***, obteniendo las siguientes salidas:

Ejecución del comando en la centralita PBX-A (en esta centralita se recuerda que está registrada la extensión 201, por lo que debemos preguntar por la extensión 200)

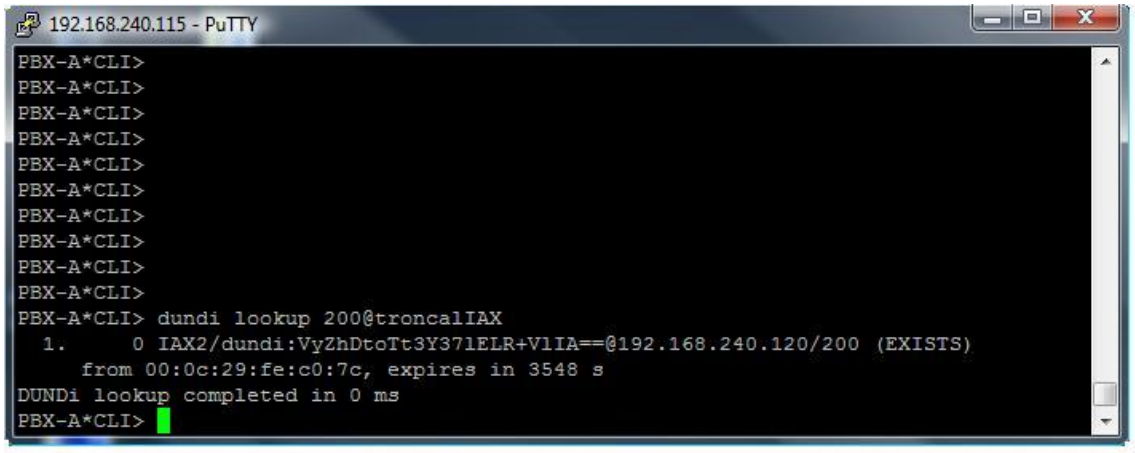

Figura 27: Ejecución del comando *dundi lookup* PBX-A

Se aprecia claramente como la consulta ha sido realizada de forma satisfactoria puesto que nos ha aportado la cadena de llamada hacia la extensión 200. Como se podrá ver en el ANEXO en la configuración del archivo dundi.conf, esta cadena corresponde con la configurada en la sección de [mappings], donde se indica el tipo de canal, IAX2, el nombre del usuario para este canal, dundi, la contraseña de envío y por último la dirección IP de la centralita donde se encuentra registrada la extensión.

Ejecución del comando en la centralita PBX-B (en esta ocasión está registrada la extensión 200, por lo que debemos preguntar por la extensión 201)

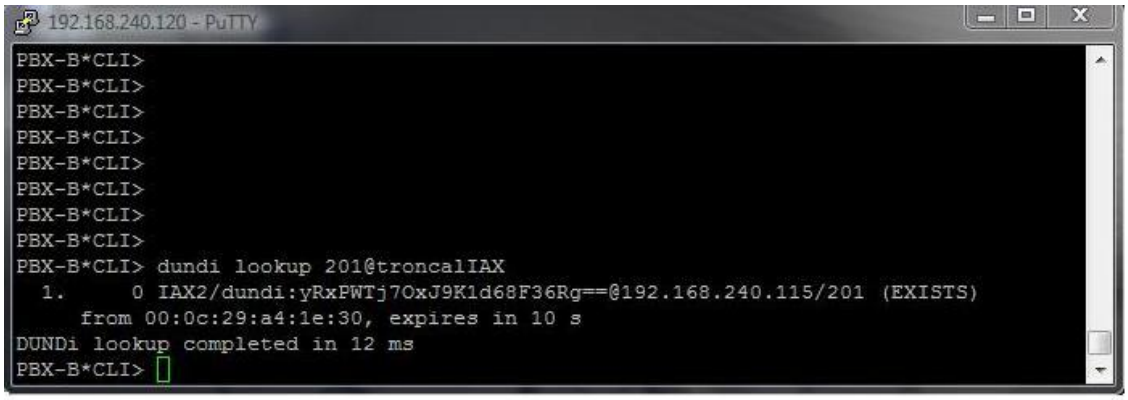

Figura 28: Ejecución del comando *dundi lookup* PBX-B

Al igual que en la anterior consulta, ésta se ha realizado de forma satisfactoria, obteniendo la cadena de llamada a la extensión 201. Se pueden apreciar los mismos detalles explicados en el anterior ejemplo: canal, nombre del usuario, contraseña y dirección IP de la centralita donde reside la extensión.

Después de realizar todas estas pruebas, se puede afirmar que nuestra configuración DUNDi ha sido un éxito puesto que es capaz de intercambiar información de las extensiones registradas en cada centralita. Ahora, si la llamada falla, sabemos que debe ser a causa del dialplan en RealTime y no de la configuración de DUNDi, por tanto, es momento de llamar de una extensión a otra.

En las siguientes imágenes se van a mostrar dos llamadas con su correspondiente ejecución en la centralita. Vamos a comprobar que efectivamente toda la configuración tanto de Realtime como de de DUNDi funciona correctamente.

Llamada de la extensión 200 a la 201:

En primer lugar observamos qué ocurre en la centralita 192.168.240.120 que es donde se encuentra registrada la extensión 200 (extensión origen):

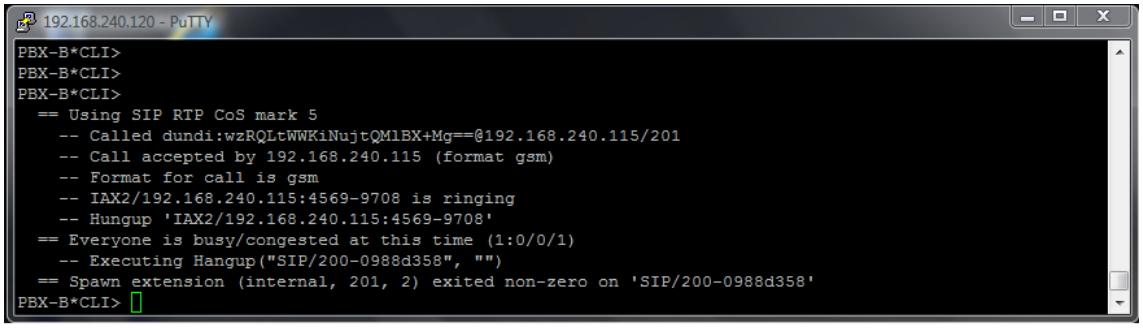

Figura 29: Llamada DUNDi centralita origen PBX-B

Se aprecia muy claramente la ejecución de la llamada. Como se sabe, ya que hemos forzado la prueba a ello, las extensiones están en centralitas diferentes, por lo que realiza una llamada DUNDi, utilizando el formato que se definió en la etiqueta [mappings] del fichero de configuración dundi.conf. La llamada es aceptada por la otra centralita, es decir, va a tramitar la llamada, no quiere decir que la llamada haya sido aceptada por la extensión destino. Aplican un formato de llamada GSM y crean un canal IAX2 entre ambas centralitas. Por último, ahora sí, el usuario con extensión 201 rechaza la llamada.

Ahora veremos que ocurría en la centralita donde se encontraba registrada la extensión 201:

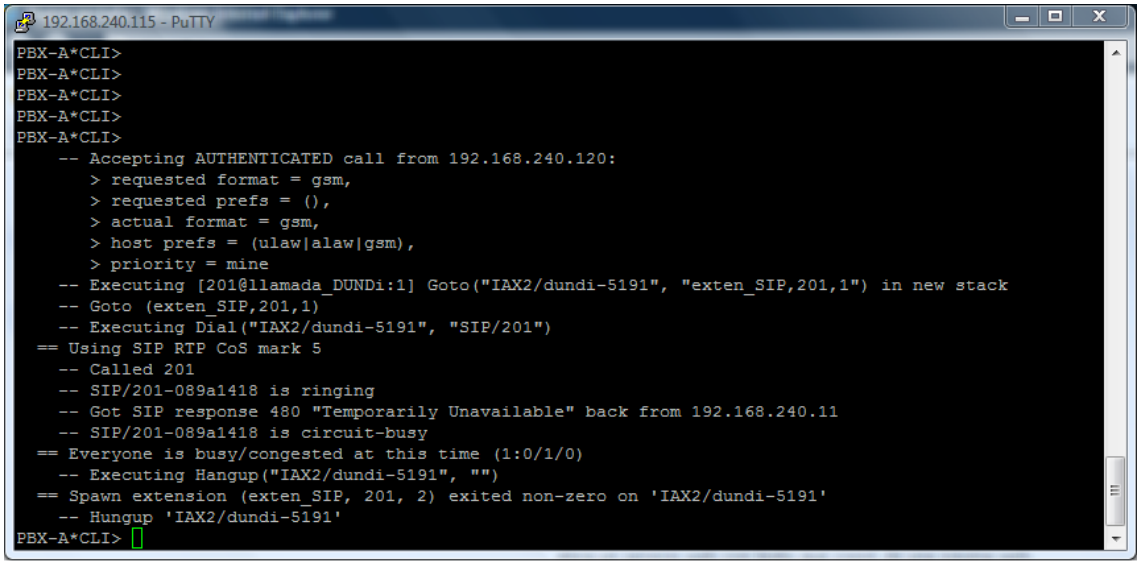

Figura 30: Llamada DUNDi centralita destino PBX-A

En primer lugar se observa que debe aceptar una llamada autenticada desde una centralita remota, lo que nos hace indicar que se trata de una llamada DUNDi. Una vez aceptada, negocian el formato de la llamada para posteriormente comenzar con la ejecución de los contextos de llamada. Si se recuerda el orden en que deberían ejecutarse los contextos para una llamada DUNDi es fácil darse cuenta que la llamada es correcta. Primero ejecuta el contexto "llamada_DUNDi", puesto que al crear el canal IAX2 la llamada era enviada a dicho contexto. Seguidamente, "llamada_DUNDi" redirige la llamada al contexto "exten_SIP" que es ahí donde se encuentran las reglas de

marcado entre extensiones. Por último, se puede apreciar cómo se acepta la llamada hacia la extensión 201 como si fuera una llamada interna (gestionada por la misma centralita) y como ésta rechaza la llamada.

A continuación se muestran las imágenes de la llamada inversa, es decir, de la extensión 201 a la 200, para ver que en sentido contrario también funciona. Se omite explicación alguna puesto que es exactamente igual que en el anterior ejemplo. Sólo resaltar que en esta ocasión la llamada es aceptada y posteriormente cortada.

Imagen de la centralita origen de la llamada (extensión 201):

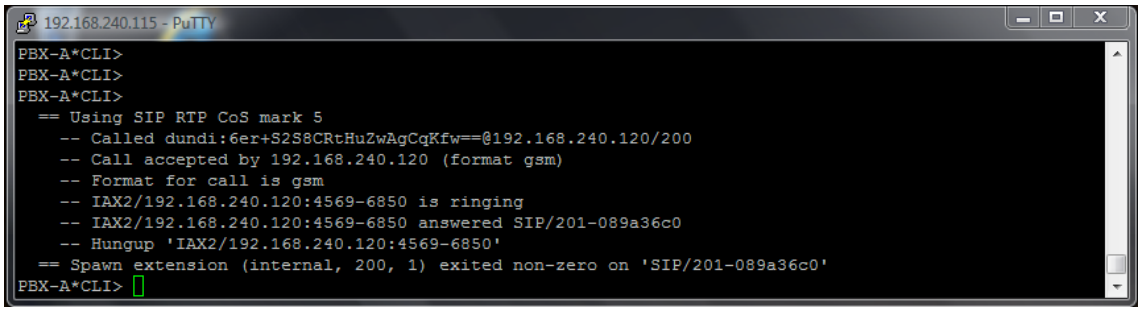

Figura 31: Llamada DUNDi centralita origen PBX-A

Imagen de la centralita destino de la llamada (extensión 200):

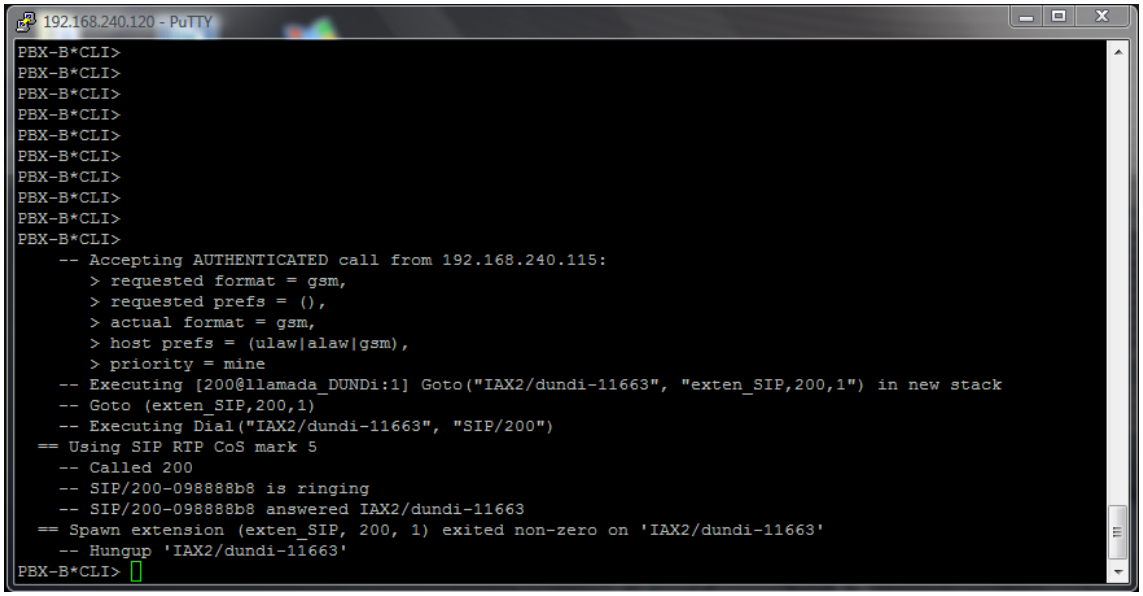

Figura 32: Llamada DUNDi centralita destino PBX-B

Con todo lo realizado hasta el momento, se puede afirmar que nuestra red de alta disponibilidad está preparada para montarse en un entorno de pruebas más sofisticado que unas simples máquinas virtuales. Dichas pruebas se realizarán en la empresa Intecdom y en la presentación del proyecto.

Capítulo 6

Planificación. Memoria Económica del Proyecto

A lo largo de este capítulo se pretende dar una visión económica en lo que a presupuesto se refiere. Como ya se explicó al inicio de este documento en el primer capítulo, se pretende exponer el coste que ha implicado realizar este proyecto fin de carrera. En una segunda visión se explicará una solución como posible producto en el mercado y el coste que éste implicaría. También se mostrará una estimación aproximada de las diferentes tareas llevadas a cabo y su duración.

6.1 Estimación de la planificación de tareas

En el siguiente apartado, se muestra una tabla con la estimación de la duración de las distintas tareas a realizar para la consecución de este proyecto.

Como se puede apreciar, el gráfico representa una estimación aproximada por semanas.

Semanas / Tareas	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A														
B														
C														
D														
E														
F														

Tabla 1: Planificación de tareas

Relación de Tareas:

A: Búsqueda de información
B: Toma de contacto con la tecnología
C: Ideas de desarrollo y diseño de la aplicación
D: Desarrollo de la aplicación
E: Pruebas
F: Redacción de la memoria

6.2 Coste de la realización del proyecto

A continuación se muestra el coste personal que ha implicado la realización del proyecto fin de carrera

1) Ejecución Material

- Ordenador portátil…… 599 €
- Consumo del ordenador personal……..... 150 €
- Material de oficina..... 50 €
- Total de ejecución material..... 799 €

2) Material fungible

- Gastos de impresión..... 50 €
- Encuadernación.....10 €

3) Total presupuesto

- Total Presupuesto..... 859 €

6.3 Posible venta del proyecto en el mercado

Ya sabemos lo que ha costado realizar el proyecto, ahora intentemos saber lo que costaría llevarlo a cabo para venderlo como producto en el mercado. Los datos aportados en base al número de máquinas necesarias no tendrán que ver con la realización de este proyecto fin de carrera, puesto que para éste se han utilizado máquinas virtuales.

Lo primero de todo es concretar las máquinas necesarias para la implementación de la red. Se va a suponer un clúster de dos centralitas Asterisk, que sería el mínimo de centralitas para formarlo. Por tanto, se necesitarán las siguientes máquinas:

- 2 equipos para servidores DNS
- 2 equipos para centralitas Asterisk
- 1 equipo para servidor web

En cuanto a características de los equipos, se pueden barajar diferentes opciones. Si la implementación no va a ser excesivamente profesional, como por ejemplo, para un entorno de pruebas, o para una pequeña oficina, podría valer cualquier equipo PC superior a un Pentium IV y al menos 1GB de RAM, para cualquiera de los tres servidores que necesitamos. En cambio, el estudio realizado a continuación es para entornos profesionales, donde se disponga de una torre de servidores tipo rack para su colocación. De esta forma, se ha realizado una configuración de servidor DELL como posible solución que se muestra a continuación:

Servidor básico: **PowerEdge R200**

Figura 33: Servidor PowerEdge R200

Posible configuración orientativa respecto a este servidor:

- **Procesador:** Quad Core Intel® Xeon® X3323, 2.5GHz, 2x3M Cache, 1333MHz FSB
- **Memoria:** 2GB DDR2 667MHz Memory - 1CPU (2x1GB single rank DIMMs).
- **Primera unidad de disco duro:** 250GB, SATA, 3.5-inch, 7.200 rpm Hard Drive.
- **Segunda unidad de disco duro:** 160GB, SATA, 3.5-inch, 7.200 rpm Hard Drive (pensada para posibles copias de seguridad, aunque es preferible en disco externo).
- **Sistema operativo instalado en fábrica:** Ninguno.
- **Software de backup:** Ninguno.
- **Unidad óptica:** Internal SATA CD-RW/DVD-ROM Drive.
- **2ª tarjeta de red de respaldo:** Broadcom® NetXtreme 5722 Single Port Gigabit Ethernet NIC, PCIe.

Como se puede observar, los requisitos son algo superiores a lo que realmente necesita cualquiera de los tres servidores, pero la elección ha sido esta de cara a que no se quede obsoleto en un futuro. Es seguro que Asterisk evolucionará, y con ello, los requisitos mínimos de los sistemas (comenzó con un Pentium III y ya se aconseja mínimo Pentium IV con 1 GB de RAM). En cuanto al servidor web, puede alojar contenido de otras aplicaciones web, incluso dependiendo del tipo de empresa, se podría ofrecer la posibilidad de realizar Housing y rentabilizar el servidor o incluso los servidores.

Si se ha observado en la descripción de los servidores, éstos no dispondrían de sistema operativo, por una razón muy concreta. Los servidores a utilizar, tanto DNS, Asterisk y Tomcat, corren sobre sistemas Linux, software libre y completamente gratuito, por tanto, no tenemos ningún coste en cuanto a sistemas operativos ni aplicaciones se refiere. Esto quiere decir que se utilizaría como sistema operativo

Debian (el mismo utilizado en las máquinas virtuales del proyecto) y como aplicaciones exactamente las mismas que en el proyecto, BIND 9 para los servidores DNS, Asterisk 1.6.1.1 para las centralitas, MYSQL para las bases de datos y Tomcat para el servidor web.

A continuación se muestra una tabla orientativa con el coste total:

Referencia dispositivo	Fabricante	Descripción	Precio/ Unidad	Unidades	Precio total
Servidor Dell R200	Dell	Servidor Tipo Rack	499 €	5	2495 €
Switch	Netgear	Switch de 8 puertos para interconexión de máquinas	29,90 €	1	29,90 €
Sistema operativo Debian	Debian Project	Sistema operativo Debian para los servidores	gratuito	5	0 €
Software Asterisk	Digium	Software necesario para las centralitas	gratuito	2	0 €
Bind 9	-	Software necesario para los servidores DNS	gratuito	2	0 €
Bases de datos MySQL	Sun MySQL	Istema de gestión de bases de datos MySQL	gratuito	2	0 €
Apache Tomcat	Fundación Apache	Servidor de aplicaciones web	gratuito	1	0 €
TOTAL					**2524,9 €**

Tabla 2: Coste material del proyecto

Ahora hay que contemplar el salario del ingeniero encargado de llevar a cabo el proyecto, en este caso Ingeniero Técnico de Telecomunicación especialidad Telemática. Supongamos el mismo número de horas invertidas para realizar este proyecto fin de carrera, lo que nos da un total de aproximadamente 300 horas. De este modo, el coste del ingeniero se elevaría a:

1) Sueldo Ingeniero

- 300horas/10€ hora…… 3000 €

Por tanto, con el gasto material y el sueldo del ingeniero técnico la cifra total y por tanto el presupuesto estimado para la realización de este proyecto fin de carrera sería:

- **Presupuesto Total**……... 5524,90 €

Ahora sólo queda concretar la forma de explotar el proyecto en el mercado. Supongamos una consultoría dedicada a Asterisk. Ésta puede ofrecerles el producto a sus clientes de formas distintas. Una primera posibilidad sería cobrarles el servicio haciendo uso de nuestros propios servidores para su red de telefonía, de forma que rentabilicemos el gasto en los servidores y además se ganara dinero con ello. O bien, se puede ofrecer una instalación completa con servidores incluidos que fueran gestionados por el propio administrador del cliente. Son meras suposiciones, con la intención de rentabilizar el gasto en el proyecto y sacar provecho de ello.

Capítulo 7

Análisis de Futuro

Como todo proyecto, es lógico pensar que se podría llevar a cabo una nueva línea de desarrollo para el producto y en este caso no podría ser menos. Si se ha entendido bien el funcionamiento del clúster de Asterisk, se aprecian ciertas mejoras de cara a la administración del mismo. Como se ha explicado a lo largo del documento, el clúster de Asterisk está montado en RealTime, lo que conlleva disponer de una base de datos con información de interés para la gestión de la centralita. Pero esta tecnología sólo está disponible para unos módulos en concreto, y entre ellos no figura la configuración de DUNDi en RealTime. Esto sería una buena estrategia de futuro porque facilitaría la administración de la centralita de forma plena con nuestro sistema de supervisión. A continuación se detalla el por qué:

Una vez se tiene montado el clúster de Asterisk, no sería de extrañar que en un futuro se quisiera ampliar el número de centralitas que formaran dicho clúster. De ser así, siempre que incluimos una centralita, debemos acceder a los ficheros de configuración, en concreto a dundi.conf e iax.conf, para establecer el anuncio de extensiones y crear el canal de comunicación con la nueva centralita. Esto conllevaría dejar inutilizadas las centralitas durante el tiempo de actualización de las mismas. Por tanto, una buena opción sería obtener la información de la red DUNDi desde la base de datos, de forma que podamos modificar y actualizar las centralitas sin necesidad de que éstas paren el servicio telefónico. Si esto fuera posible, bastaría con crear una nueva aplicación en el sistema de supervisión para que del mismo modo que gestionaba las extensiones y el dialplan, lo hiciera con la red DUNDi.

En base al sistema de supervisión, se ha observado que sólo se avisa al administrador cuando la base de datos no está disponible, no siendo así para la centralita, es decir, que la base de datos esté inaccesible no implica que también lo esté Asterisk. Esto se debe a que no debe preocuparnos que la centralita quede inaccesible, ya que cuando esto ocurre, automáticamente los terminales buscarán una nueva centralita operativa y seguirán funcionando sin problema. En cambio, no ocurre esto cuando la base de datos está fuera de servicio. Si esto ocurre, la centralita que gestiona la base de datos inaccesible queda inutilizada, es decir, de cara a los terminales sigue funcionando pero no podrá gestionar llamadas, realizar nuevos registros…y todo esto sin que los terminales puedan registrarse contra otra centralita. Es por esta razón por la cual sólo se avisa de la caída de las bases de datos. Aún así, una buena funcionalidad de cara a un futuro para este sistema de supervisión sería programar una pequeña aplicación con el API Asterisk-Java que nos permita saber si las centralitas están operativas, exactamente igual que ahora con las bases de datos.

Otro aspecto a tener en cuenta es conectar nuestra centralita a la red tradicional o PSTN para poder cursar llamadas también por parte de la red analógica. A lo largo del proyecto, sólo hemos tenido en cuenta la necesidad de crear el clúster de Asterisk pensando sólo en llamadas internas a la organización, en ningún momento se ha barajado la posibilidad de conectar las centralitas a la red tradicional. Si tenemos en cuenta este aspecto, debemos ser conscientes del número de líneas disponibles por parte de la organización.

Bibliografía

[1] Van Meggelen J., Smith J., Madsen L.; *Asterisk. The Future of Telephony. Ed. O'Reilly (2005)* ISBN: 0-596-00962-3.

[2] Andrew S. Tanenbaum; *Redes de computadoras. PEARSON EDUCACIÓN. Cuarta Edición (2003)* ISBN: 970-26-0162-2

[3] José Manuel Huidobro Moya, Rafael Conesa Pastor; *Sistemas de TELEFONÍA. THOMSON-PARANINFO (2006)* ISBN: 84-283-2927-3

[4] Daniel L. Morril; *Configuración de sistemas LINUX. Grupo Anaya (2002)* ISBN: 84-415-1465-8

[5] Web Asterisk: http://www.asterisk.org

[6] Sugerencias sobre configuración Asterisk: http://www.voipnovatos.es/

[7] Comunidad de usuarios Asterisk España: http://www.asterisk-es.org

[8] Información acerca de codecs y protocolos para VoIP: http://www.voipforo.com/

[9] Protocolo DUNDi: http://www.dundi.com/

[10] Información general Wikipedia: http://wikipedia.org

ANEXOS

Anexo A. Protocolos VoIP

1.- Protocolo H.323

En el siguiente Anexo se pretende dar una breve descripción acerca del protocolo H.323. No se profundiza puesto que el protocolo que se utiliza en el proyecto es SIP, que veremos en el siguiente apartado.

Esta norma data de 1996 (versión 1) y 1998 (versión 2). Ha sido generada para sistema de comunicación multimedia basado en paquetes (redes que pueden no garantizar correctamente la calidad de servicio QoS). Esta tecnología permite la transmisión en tiempo real de vídeo y audio por una red de conmutación de paquetes, como puede ser Internet. Es de suma importancia ya que los primeros servicios de voz sobre protocolo internet (VoIP) utilizaban esta norma. En la versión 1 del protocolo H.323 se disponía de un servicio con calidad de servicio no garantizada sobre redes LAN. En la versión 2 se definió la aplicación VoIP independiente del medio. La versión 3 de 1999 incluye el servicio de fax sobre IP (FoIP) y conexiones rápidas entre otros, y en 2003 apareció la última versión, la v5.

La versión 2, muy extendida, introduce una serie de mejoras sobre la primera versión. Algunas de ellas son:

- Permite la conexión rápida (elimina parte de tiempo de solicitud de conexión).
- Mediante H.235 introduce funciones de seguridad (autenticación, integridad, privacidad).
- Mediante H.450 introduce los servicios suplementarios.
- Soporta direcciones del tipo RFC-822 (e-mail) y del formato URL.
- Mediante una unidad MCU permite el control de llamadas multi-punto (conferencia entre varios interlocutores).
- Permite la redundancia de gatekeepers.
- Soporta la codificación de vídeo en formato H.263.
- Admite mensajes RIP (Request in Progress) para informar que la llamada no puede ser procesada por el momento.
- Provee la facilidad de que el Gateway informe al gatekeeper sobre la disponibilidad de enlaces para mejorar el enrutamiento de llamadas.

2.- Protocolo SIP

El IETF (Internet Engineering Task Force) ha generado un set de protocolos que simplifican notablemente las funciones del H.323, el cual tiene previstas funciones dentro de una red corporativa y multimedia. SIP (Session Initial Protocol) RFC 2543 es un protocolo más simple que H.323 y está basado en HTTP adoptando las características más importantes de este estándar como son la sencillez de su sintaxis y una estructura cliente-servidor basada en un modelo petición-respuesta. Como se ha visto anteriormente, en H.323 se utiliza el Gatekeeper mientras que en SIP se usa el SIP-Server, el cual tiene mejores aspectos de escalabilidad para grandes redes.

SIP es un protocolo sencillo y extensible, basado en texto y el mensaje basado en HTTP (RFC-2068 para la semántica y sintaxis). La dirección usada en SIP se basa en un localizador URL (Uniform Resource Locator) con un formato del tipo dirección de correo electrónico, user@domain.com integrando así su servicio en Internet. En este modelo se requiere de la ayuda de un servidor de resolución de nombres, DNS (Domain Name Server).

El protocolo SIP incorpora también funciones de seguridad y autenticación, así como la descripción del medio mediante el protocolo SDP. Para el proceso de facturación se puede recurrir a un servidor RADIUS.

Las fases de comunicación soportadas en una conexión unicast mediante el protocolo SIP son las siguientes:

- **User location:** En esta fase se determina el sistema terminal para la comunicación.
- **User capabilities**: Permite determinar los parámetros del medio a ser usados.
- **User availability**: Para determinar la disponibilidad del llamado para la comunicación.
- **Call setup:** Para el establecimiento de la llamada entre ambos extremos.
- **Call handing**: Incluye transferencia y terminación de la llamada.

El protocolo SIP tiene dos tipos de mensajes: Request y Response. El mensaje de Request es emitido desde el cliente terminal al servidor terminal. El encabezado de los mensajes request y response contiene campos similares:

- **Start Line**: Usado para indicar el tipo de paquete, la dirección y la versión de SIP.
- **General Header**: Contiene el Call-id (se genera en cada llamada para identificar a la misma); Cseq (se inicia en un número aleatorio e identifica de forma secuencial a cada request); From (es la dirección del origen de la llamada); To (es la dirección del destino de la llamada); Via (sirve para recordar la ruta del

request; por ello cada proxy en la ruta añade una línea de via) y Encryption (identifica un mensaje que ha sido cifrado para seguridad).

- **Additionals**: Además del encabezado general, se pueden transportar campos adicionales. Por ejemplo: Expire indica el tiempo de validez del registro; Priority indica la prioridad del mensaje, etc.

Se han definido seis métodos para los mensajes de request-response:

- **Invite**: para invitar al usuario a realizar una conexión. Localiza e identifica al usuario.

- **Bye**: Para la terminación de una llamada entre usuarios.

- **Options**: Información de capacidades que pueden ser configuradas entre agentes o mediante un servidor SIP.

- **ACK**: Usado para reconocer que el mensaje Invite puede ser aceptado.

- **Cancel**: Termina una búsqueda de un usuario.

- **Register**: Emitido en un mensaje multicast para localizar al server SIP.

La arquitectura del protocolo SIP está integrada en la infraestructura web y sigue básicamente un modelo cliente-servidor, reutilizando conceptos de otros servicios (web, correo, dns). Los principales elementos que constituyen la arquitectura de este protocolo son:

- Agentes de usuario:
  - Agentes de usuario clientes (UAC).
  - Agentes de usuario servidores (UAS).
- Servidores:
  - Proxys, de registro y de redirección.

Un Proxy Servidor es una aplicación intermedia que actúa tanto como servidor y cliente, generando mensajes SIP a nombre del cliente que generó el mensaje original.

- Los mensajes pueden ser respondidos o encaminados a otros servidores donde se encuentra la localización actual del usuario

  - Interpreta, re-escribe o traduce los mensajes antes de encaminarlos.
  - Autentifica y autoriza a los usuarios para los servicios
  - Implementa políticas de encaminamiento.
  - Hay dos tipos de Proxy Server:

- Outbound Proxy: Permite simplificar la administración de los usuarios de un dominio, aplica políticas, tarifica, etc.

- Inbound Proxy: Permite independizar al usuario del dispositivo que utiliza y de su localización.

Un mismo servidor puede funcionar como Proxy saliente o entrante de un dominio.

Las invitaciones de SIP son usadas para crear sesiones y llevan las descripciones de la sesión que permiten que los participantes convengan en un sistema de tipos de medios.

Un Servidor de redirección es un agente de usuario servidor (UAS) que genera respuestas a las peticiones que recibe, ordenando al cliente entrar en contacto con un sistema alterno de URIs. En algunas arquitecturas puede ser deseable reducir la carga de los servidores Proxy que son responsables de las peticiones de encaminamiento, y mejorar la robustez del recorrido de los mensajes de señalización, mediante redirecciones.

La redirección permite que los servidores envíen la información de encaminamiento para una petición como respuesta al cliente, de tal modo quitándose del camino de los subsiguientes mensajes para una transacción mientras que ayudan en la localización del blanco de la petición. Cuando el autor de la petición recibe el cambio de dirección, enviará una nueva petición basada en la URI (s) que ha recibido. Propagando URIs desde el núcleo de la red hacia sus extremos, el cambio de dirección permite obtener una escalabilidad considerable en la red.

Un servidor de registro es un servidor que acepta peticiones de tipo REGISTER y pone la información que recibe de esas peticiones en el servicio de localización para el dominio que maneja (IP + Puerto) ya que los usuarios pueden tener múltiples localizaciones. Normalmente es el mismo servidor que el Proxy.

El protocolo SIP permite implementar dos tipos de movilidad diferentes:

- La movilidad personal donde el usuario puede ser alcanzado en un dispositivo cualquiera, registrándose en el servidor de registro.

- La movilidad propia al protocolo IP (VPN).

El registro permite mantener las localizaciones actuales del usuario de manera dinámica. Se basa en la localización actual. El servidor Proxy encaminará las llamadas al destino. SIP ofrece la capacidad de descubrimiento. Si un usuario desea iniciar una sesión con otro usuario, el SIP debe descubrir el host actual en el cual el usuario de destinación es accesible. Este proceso de descubrimiento es logrado por elementos de la red SIP tales como servidores proxy y servidores de redirección que son los responsables de recibir una petición, determinar dónde enviarla basados en el conocimiento de localización de usuarios, y después enviarla allí. Para hacer esto, los elementos de la red SIP consultan un servicio abstracto conocido como servicio de localización, que proporciona los mapeos de dirección para un dominio particular. Éstos mapeos de direcciones mapean SIP URIs entrantes.

Las principales desventajas que conlleva el uso de este protocolo son:

- Problemas de Red: La utilización de un canal PtP para el streaming de audio RTP plantea numerosos problemas a nivel de red: NAT routers, firewall, etc.

- Interoperabilidad con PSTN: El protocolo H.323 ofrece mayores ventajas.

Anexo B. Codecs

Los codec o codificadores de audio se utilizan para digitalizar, comprimir y codificar la señal de audio analógica para poder ser transmitida por la red IP. Los codecs son algoritmos matemáticos implementados en software. Existen diversos modelos utilizados en VoIP dependiendo del algoritmo escogido en la transmisión, la calidad de la voz, el ancho de banda necesario y la carga computacional. El principal objetivo es aunar la eficiencia y la calidad de la voz. El sistema auditivo del ser humano es capaz de captar las frecuencias comprendidas entre 20 Hz y 20 kHz y la mayoría de codecs procesan la información dentro de la banda de 400 Hz- 3,5 kHz para que a la hora de reconstruir la señal, ésta siga siendo inteligible.

Entre los codecs más comunes se encuentran:

- **G.711:** Estándar de la ITU-T para la compresión de audio para telefonía. Representa las señales de audio mediante muestras comprimidas en una señal digital con tasa de muestreo de 8000 muestras por segundo con un flujo de datos de 64 Kbps. El algoritmo es logarítmico y existen dos leyes:

- **μ-law:** Usado sobre todo en Norte América y Japón. Codifica cada 14 muestras en palabras de 8 bits.

- **a-law:** Usado en Europa y en el resto del mundo. Codifica cada 13 muestras en palabras de 8 bits

- **G.723:** Algoritmo estandarizado por la ITU-T en 1995 puede operar a 6.3 Kbps o 5.3 Kbps.

- **G.726:** Estándar de la ITU-T, conocido también como ADPCM (Adaptative Differential Pulse Code Modulation), sustituyó a G.721 en 1990. Permite trabajar con velocidades de 16, 24, 32 y 40 Kbps. Este codec proporciona una disminución considerable del ancho de banda sin aumentar en gran medida la carga computacional.

- **G.729:** Se usa sobre todo en aplicaciones de Voz sobre IP por los bajos requerimientos en ancho de banda. Opera con tasas de 8 Kbps pero existen extensiones para tasas de 6.4 y 11.8 Kbps para peor o mejor calidad de voz respectivamente.

- **GSM** (Global System Mobile): Estándar que opera a 13 Kbps con una carga de CPU aceptable. Inicialmente fue desarrollado para la telefonía móvil.

- **iLBC** (Internet Low Bit rate Codec): Algoritmo complejo desarrollado por Global IP Sound (GIPS) que ofrece una buena relación ancho de banda/calidad de voz a cambio de una mayor carga computacional. Opera a 13.3 Kbps y 15.2 Kbps.

- **Speex:** Implementa un algoritmo capaz de variar la velocidad de transmisión dependiendo de las condiciones actuales de la red (VBR: Variable Bit Rate). El ancho de banda puede variar desde 2.15 a 22.4 Kbps.

- **MP3** (Moving Picture Experts Group Audio Layer 3 Enconding Standard): Codec de audio optimizado para música y no para telefonía creado por la ISO. Es utilizado sobre todo en los teléfonos IP para la música en espera.

Anexo C. DNS

Como ya se sabe, el DNS (Domain Name System) tiene como función principal la resolución de nombres en direcciones IP. En este Anexo se pretende dar una visión acerca de qué es DNS y como funciona su jerarquía.

Hace años, en los tiempos de ARPANET, sólo había un archivo, host.txt, en el que se listaban todos los host y sus direcciones IP. Cada noche, todos los hosts obtenían este archivo del sitio en el que se mantenía. En una red conformada por unas cuantas máquinas grandes de tiempo compartido, este método funcionaba razonablemente bien.

Sin embargo, cuando miles de estaciones de trabajo se conectaron a la red, todos se dieron cuenta de que este método no podría funcionar eternamente. Por una parte, el tamaño del archivo crecería de manera considerable. Un problema aún más importante era que ocurrirían conflictos constantes con los nombre de los hosts a menos de que dichos nombres se administraran centralmente, algo impensable en una red internacional de grandes dimensiones. Para resolver este problema se inventó el DNS.

La esencia del DNS es la invención de un esquema de nombres jerárquico basado en dominios y un sistema de base de datos distribuido para implementar este esquema de nombres. El DNS se usa principalmente para relacionar los nombres de host y destinos de correo electrónico con las direcciones IP, pero también puede usarse con otros fines. El DNS se refiere en los RFCs 1034 y 1035.

Muy brevemente, la forma en que se utiliza el DNS es la siguiente. Para relacionar un nombre con una dirección IP, un programa de aplicación llama a un procedimiento de biblioteca llamado resolvedor (más conocido como resolver), y le pasa el nombre como parámetro. El resolver envía un paquete UDP a un servidor DNS local, que después busca el nombre y devuelve la dirección IP al resolver, que entonces lo devuelve al solicitante. Una vez que tiene la dirección IP, el programa puede establecer una conexión TCP con el destino, o enviarle paquetes UDP.

Conceptualmente, Internet se divide en 200 dominios de nivel superior, cada uno de los cuales abarca muchos hosts. Cada dominio se divide en subdominios, los cuales, a su vez, también se dividen, y así sucesivamente. Todos estos dominios pueden representarse mediante un árbol. Las hojas del árbol representan los dominios que no tienen subdominios (pero que evidentemente contienen máquinas). Un dominio de hoja puede contener un solo host, o puede representar a una compañía y contener miles de host.

Los dominios de nivel superior se dividen en dos categorías: genéricos y de país. Los dominios genéricos originales son com (comercial), edu (instituciones educativas), gov (el gobierno federal de Estado Unidos), int (ciertas organizaciones internacionales), mil (fuerzas armadas de estados Unidos), net (proveedores de red) y org (organizaciones no lucrativas). Los dominios de país incluyen una entrada para cada país, como se define en la ISO 3166. Éstos, como cabe de imaginar, son del tipo es (España), it (Italia), fr (Francia)… En la imagen siguiente se puede apreciar una estructura jerarquizada de lo dicho hasta ahora. Como se apreciará en la imagen, existe un dominio especial, llamado ARPA. Este dominio es utilizado para la resolución inversa de direcciones, es decir, permite obtener el nombre de una máquina a partir de su dirección IP.

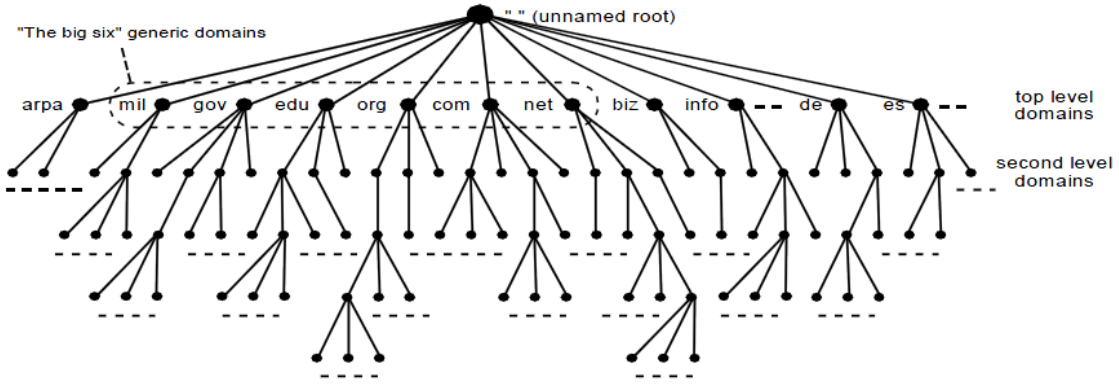

Figura 34: Esquema jerárquico de DNS

Aunque se ha dicho que son siete los dominios de propósito general, lo cierto es que en el año 2000, ICANN (Internet Corporation for Assigned Names and Numbers) aprobó cuatro nuevos dominios de nivel superior: biz (negocios), info (información), name (nombres de personas) y pro (profesiones).

Cada dominio se nombra por la ruta hacia arriba desde él a la raíz. Los componentes se separan por puntos. Así, se puede obtener el dominio it.uc3m.es perteneciente al departamento de Ingeniería Telemática de la universidad Carlos III. Los nombres de dominio no hacen distinción entre mayúsculas y minúsculas, teniendo además un máximo de 255 caracteres y de 63 caracteres por componente.

Cabe resaltar un pequeño detalle. Los nombres reflejan los límites organizacionales, no las redes físicas. Por ejemplo, si los departamentos de informática e ingeniería telemática se ubican en el mismo edificio y comparten la misma LAN, pueden tener dominios diferentes. De igual modo, si el departamento de Ingeniería Telemática se divide en varios edificios, todos los hosts de ambos edificios pertenecerán, por lo general, al mismo dominio.

Cada dominio, sea un host individual o un dominio de nivel superior, puede tener un grupo de registros de recursos asociados a él. En un host individual, el registro de recursos más común es simplemente su dirección IP, pero también existen muchos otros tipos de registros de recursos. Cuando un resolver da un nombre de dominio al DNS, lo que recibe son los registros de recurso asociados a ese nombre. Por lo tanto, la función real del DNS es relacionar los dominios de nombres con los registros de recursos.

Un registro de recurso tiene cinco tuplas. Aunque éstas se codifican en binario por cuestión de eficiencia, en la mayoría de las presentaciones, los registros de recursos se dan como texto ASCII, una línea por registro de recursos. El formato habitual que se usa es el siguiente:

Nombre_dominio Tiempo_de_vida Clase Tipo Valor

El Nombre_dominio indica el dominio al que pertenece el registro de recurso. Por lo general, existen muchos registros por dominio y cada copia de la base de datos contiene información de muchos dominios. Por lo tanto, este campo es la clave primaria

de búsqueda usada para atender las consultas. El orden de los registros de la base de datos no es significativo.

El campo de Tiempo_de_vida es una indicación de la estabilidad del registro. La información altamente estable recibe un valor grande, como 86400 (segundos que tiene un día). La información altamente volátil recibe un valor pequeño, como 60. Esto se refiere a la utilidad de la caché en los sistemas DNS. Más adelante se hablará de esto.

El tercer campo de cada registro es Clase. Para la información de Internet, siempre es IN, y básicamente es el único que se utiliza, a pesar de que existen otros códigos. El campo Tipo indica el tipo de registro de que se trata. A continuación se muestran los más utilizados:

- **Registro SOA**: (Start Of Authority) Éste registro da información de una zona del/los dominios (Servidor DNS Primario, Secundario, Administrador, Tiempo de Espera...). Después de la descripción de los registros más comunes se hablará en detalle de este registro.

- **Registro NS**: El registro de recursos NS (*Name Server*) indica los servidores de nombres autorizados para la zona. Cada zona debe contener registros indicando tanto los servidores principales como los secundarios. Por tanto, cada zona debe contener, como mínimo, un registro NS. Por otra parte, estos registros también se utilizan para indicar quiénes son los servidores de nombres con autoridad en subdominios delegados, por lo que la zona contendrá al menos un registro NS por cada subdominio que haya delegado.

- **Registro MX**: Especifica un servidor de intercambio de correo para un nombre de dominio. Puesto que un mismo dominio puede contener diferentes servidores de correo, el registro MX puede indicar un valor numérico que permite especificar el orden en que los clientes deben intentar contactar con dichos servidores de correo.

- **Registro A**: Asigna un nombre de dominio completamente cualificado (FQDN) a una dirección IP, para que los clientes puedan solicitar la dirección IP de un nombre de host dado.

- **Registro CNAME**: Crea un alias (un sinónimo) para el nombre de dominio especificado.

- **Registro PTR**: Realiza la acción contraria al registro de tipo A, es decir, asigna un nombre de dominio completamente cualificado a una dirección IP. Este tipo de recursos se utilizan en la denominada *resolución inversa.*

- **Registro HINFO**: Éste registro especifica los recursos de información del host, es decir, especifica la CPU de la máquina y el SO (sistema operativo).

- **Registro SRV:** (SRV, *SeRVice*) permiten especificar de forma genérica la ubicación de los servidores para un servicio, protocolo y dominio DNS determinados.

De todos ellos, cabe destacar el registro SOA por una razón. Cada zona contiene un registro de este tipo, denominado Inicio de Autoridad o SOA (*Start Of Authority*) al comienzo del fichero de zona. Los registros SOA incluyen los siguientes campos (sólo se incluyen los que poseen un significado específico para el tipo de registro):

- **Propietario**: nombre de dominio de la zona.
- **Tipo**: "SOA".
- **Persona responsable**: contiene la dirección de correo electrónico del responsable de la zona. En esta dirección de correo, se utiliza un punto en el lugar del símbolo "@".
- **Número de serie**: muestra el número de versión de la zona, es decir, un número que sirve de referencia a los servidores secundarios de la zona para saber cuándo deben proceder a una actualización de su base de datos de la zona (o *transferencia de zona*). Cuando el número de serie del servidor secundario sea *menor* que el número del maestro, esto significa que el maestro ha cambiado la zona, y por tanto el secundario debe solicitar al maestro una transferencia de zona. Por tanto, este número debe ser incrementado (manualmente) por el administrador de la zona cada vez que realiza un cambio en algún registro de la zona (en el servidor maestro).
- **Actualización**: muestra cada cuánto tiempo un servidor secundario debe ponerse en contacto con el maestro para comprobar si ha habido cambios en la zona.
- **Reintentos**: define el tiempo que el servidor secundario, después de enviar una solicitud de transferencia de zona, espera para obtener una respuesta del servidor maestro antes de volverlo a intentar.
- **Caducidad**: define el tiempo que el servidor secundario de la zona, después de la transferencia de zona anterior, responderá a las consultas de la zona antes de descartar la suya propia como no válida.
- **TTL mínimo**: este campo especifica el tiempo de validez (o de vida) de las respuestas "negativas" que realiza el servidor. Una respuesta negativa significa que el servidor contesta que un registro no existe en la zona.

Por último, el campo Valor. Éste puede ser un número, un nombre de dominio o una cadena ASCII. La semántica depende del tipo de registro. Un claro ejemplo de cómo se forman estos registros de recursos en relación a un fichero de zona se podrá ver en el ANEXO Ficheros de configuración, en relación al fichero de zona del dominio creado para realizar el proyecto fin de carrera.

Para evitar los problemas asociados a tener una sola fuente de información, el espacio de nombres DNS se divide en zonas no traslapantes. Una zona es una parte del "árbol" gestionada por una autoridad. Los límites de una zona los decide el administrador de esa zona, eso quiere decir que cuando delega un subdominio en otra autoridad, se crea una nueva zona. Esto ocurre sin ir más lejos en el departamento de Ingeniería Telemática de la universidad Carlos III. La autoridad del dominio uc3m.es relega el control absoluto sobre el subdominio it.uc3m.es por parte del departamento,

por lo que es competencia de este último su funcionamiento. Por lo tanto, dispondrá de un servidor DNS primario y de uno o varios servidores DNS secundarios, a ser posible fuera de la zona.

Para terminar con el funcionamiento de los servidores DNS, se explicará un poco como realizan las consultas dichos servidores. Existen tres tipos diferentes de consultas: recursivas, iterativas y mixtas. En las consultas resursivas el servidor, cuando carece de la información solicitada, va a buscarla y luego retorna la información. A continuación se muestra una imagen para su comprensión

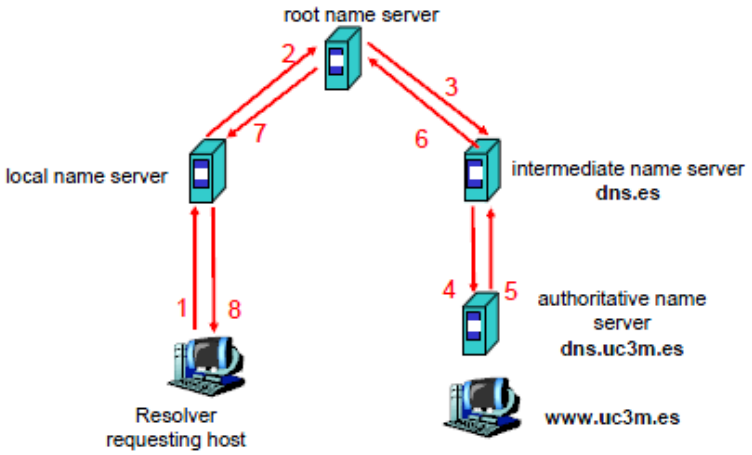

Figura 35: Consulta recursiva DNS

Las consultas iterativas, el servidor que no tiene la información solicitada devuelve el siguiente servidor a preguntar.

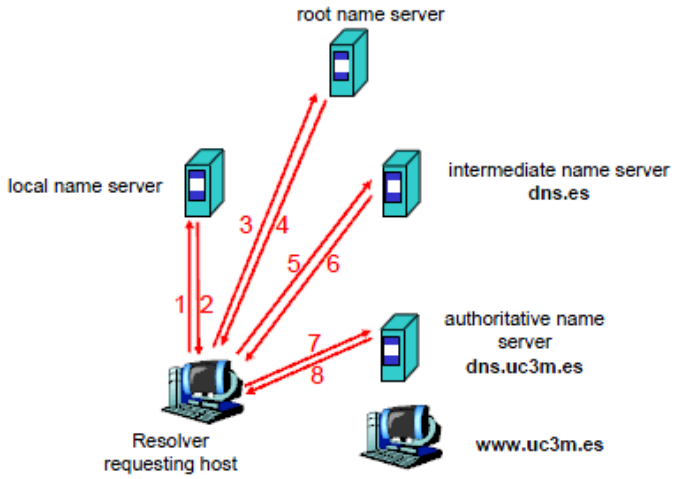

Figura 36: Consulta iterativa DNS

Por último, las consultas mixtas, como es de intuir, es una mezcla de las dos anteriores

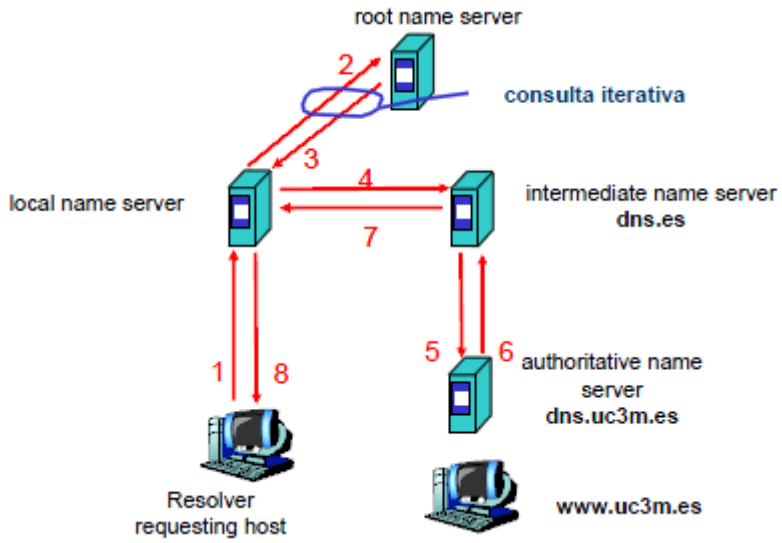

Figura 37: Consulta mixta DNS

Para reducir el tráfico, los servidores DNS intermedios suelen hacer caché de las respuestas recibidas, evitando así solicitudes continuas a servidores remotos. Entra en juego ahora el tiempo de vida (TTL). Cuando se recibe la respuesta a una consulta, ésta indica cuánto tiempo se puede almacenar en caché la información (cuánto tarda en "caducar"). Pasado este tiempo, se borra de la caché. Las respuestas obtenidas de una caché se marcan como "registro no autorizado", indicando que la información que devuelve es de "segunda mano", aunque normalmente es precisa. En ocasiones se utilizan servidores DNS que son de sólo caché (no son autorizados de ninguna zona) para reducir el tráfico DNS de un laboratorio. En la siguiente imagen se puede observar este último ejemplo:

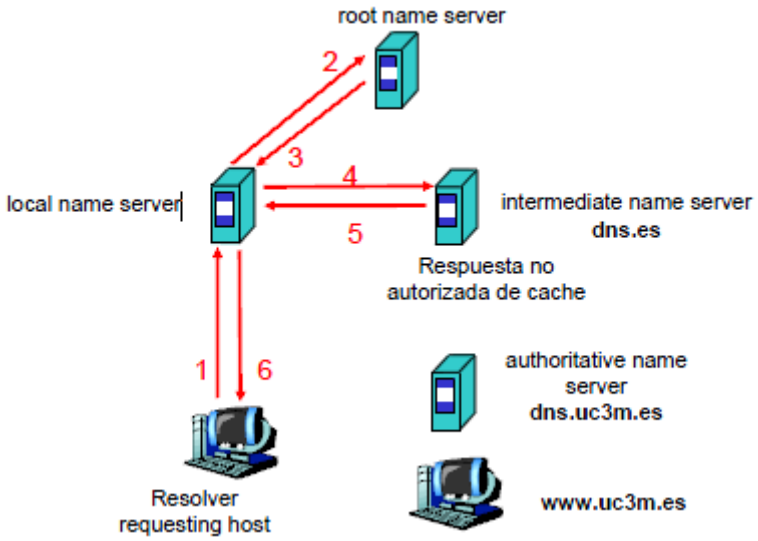

Figura 38: Esquema con servidores DNS caché

Anexo D. Instalación de Asterisk

Como se ha comentado en el capítulo 5 la versión utilizada de Asterisk ha sido la 1.6.1.1, siendo ésta una de las últimas versiones estables, que data del 18 de agosto de 2009. De igual modo, se instalará la misma versión para asterisk-addons. Este paquete hace referencia al uso de bases de datos de MySQL, entre otras cosas, ya que por defecto la instalación de Asterisk no las maneja. A continuación se muestran los elementos necesarios para la instalación llevaba a cabo en el proyecto. Para aplicaciones más avanzadas, como por ejemplo la conexión de la centralita a la red analógica, es necesario una serie de librerías que aquí no se van a explicar puesto que no forman parte del proyecto fin de carrera.

Vamos a partir de la instalación básica del sistema operativo, en este caso Debian 5.0 (Lenny). Como hay que instalar una serie de librerías para la correcta instalación de Asterisk, hay que asegurarse que los repositorios del sistema sean los correctos, si no, se pueden usar los siguientes. Simplemente hay que añadir las siguientes líneas al fichero /etc/apt/sources.list:

deb http://ftp.es.debian.org/debian/ lenny main contrib non-free
deb-src http://ftp.es.debian.org/debian/ lenny main contrib non-free

deb http://security.debian.org/ lenny/updates main contrib non-free
deb-src http://security.debian.org/ lenny/updates main contrib non-free

Por último, basta con ejecutar la siguiente sentencia para que se actualicen los repositorios:

root@PBX-A:/ # apt-get update

Ahora ya estamos preparados para instalar Asterisk.

1º) Instalamos las dependencias

root@PBX-A:/ # apt-get install linux-headers-`uname -r` build-essential cvs libnewt-dev libusb-dev libssl-dev libssl0.9.8 modconf mime-construct libxml2 libxml2-dev libtiff4 libtiff4-dev apache2 mysql-server mysql-client libmysqlclient15-dev openssl perl bison libaudiofile-dev libncurses5-dev curl sox speex libspeex-dev hdparm

A continuación bajamos los paquetes para la instalación con el comando "wget". Todo ellos se descargarán e instalarán en la carpeta /usr/src/

2º) Instalación de asterisk

root@PBX-A:/usr/src/# wget http://downloads.digium.com/pub/asterisk/releases/asterisk-1.6.1.1.tar.gz
root@PBX-A:/usr/src/# tar xzvf asterisk-1.6.1.1.tar.gz
root@PBX-A:/usr/src/# cd asterisk-1.6.1.1
root@PBX-A:/usr/src/asterisk-1.6.1.1# ./configure
root@PBX-A:/usr/src/asterisk-1.6.1.1# make menuselect (opcional)

root@PBX-A:/usr/src/asterisk-1.6.1.1# make
root@PBX-A:/usr/src/asterisk-1.6.1.1# make install
root@PBX-A:/usr/src/asterisk-1.6.1.1# make samples

3º) Instalación de asterisk-addons

root@PBX-A:/usr/src/# wget http://downloads.asterisk.org/pub/telephony/asterisk/releases/asterisk-addons-1.6.1.1.tar.gz
root@PBX-A:/usr/src/# tar xzvf asterisk-addons-1.6.1.1.tar.gz
root@PBX-A:/usr/src/# cd asterisk-addons-1.6.1.1
root@PBX-A:/usr/src/asterisk-addons-1.6.1.1/# ./configure
root@PBX-A:/usr/src/asterisk-addons-1.6.1.1/# make menuselect (opcional)
root@PBX-A:/usr/src/asterisk-addons-1.6.1.1/# make
root@PBX-A:/usr/src/asterisk-addons-1.6.1.1/# make install
root@PBX-A:/usr/src/asterisk-addons-1.6.1.1/# make samples

4º) Instalación de libiax (librería para el canal IAX entre centralitas)

root@PBX-A:/usr/src/# wget http://downloads.asterisk.org/pub/telephony/libiax/iax-0.2.1.tar.gz
root@PBX-A:/usr/src/# tar xzvf iax-0.2.1.tar.gz
root@PBX-A:/usr/src/# cd iax-0.2.1.tar.gz
root@PBX-A:/usr/src/iax-0.2.1/# ./configure
root@PBX-A:/usr/src/iax-0.2.1/# make
root@PBX-A:/usr/src/iax-0.2.1/# make install

Anexo E. Instalación y funcionamiento de Tomcat

En esta ocasión, Tomcat se instalará sobre una plataforma Windows, por lo que la instalación es sumamente simple. Lo primero de todo es instalar el JDK necesario para compilar las clases Java. Se ha utilizado la versión 1.6.0_16 y se puede adquirir desde la propia página de Sun, http://java.sun.com. Una vez instalado, tenemos que incluir dos variables de entorno en el sistema para poder compilar las clases Java. La siguiente explicación corresponde a la inserción de variables en sistemas Windows:

- Accedemos a mi PC o equipo → Propiedades de sistema → Configuración Avanzada del Sistema → Variables de entorno

- Una vez en la ventana de variables de entorno, debemos incluir en la variable PATH la siguiente sentencia: ;C:\Java\jdk1.6.0_16\bin Es muy importante comenzar con el ; para no deshabilitar la última sentencia que hubiera en la variable PATH. Esta sentencia le indica al sistema donde se encuentran los ejecutables de Java.

- Ahora debemos crear una nueva variable de entorno, propia de Java, llamada CLASSPATH y en ella alojaremos la siguiente sentencia: C:\Java\jdk1.6.0_16\lib Esta última sentencia le indica al sistema donde se encuentran las librerías necesarias para la compilación de las clases Java.

Una vez tengamos instalado el JDK, debemos instalar el JRE si no lo tuviéramos ya instalado (que es más que probable que si). Existe una gran diferencia entre el JDK y el JRE. El primero de ellos, que viene de Java Development Kit es la herramienta necesaria para poder crear aplicaciones Java, es decir, poder compilarlas y ejecutarlas. El segundo de ellos, Java Runtime Enviroment, se necesita para ejecutar aplicaciones Java desde la web, es decir, es la máquina virtual de Java. Esta última aplicación es necesaria para poder trabajar con nuestro servidor web Tomcat, ya que como cualquier otra aplicación web programada en Java, hará uso de la máquina virtual. Igualmente, se puede descargar desde la propia página de Sun, http://java.sun.com.

Ahora ya podemos proceder con la instalación de Tomcat. Se ha descargado la versión 6.18. Podemos proceder a la descarga desde la propia página del proyecto Tomcat, http://tomcat.apache.org/, y basta con seguir las indicaciones en la instalación. Simplemente nos preguntará donde está instalado en el equipo el JRE. Una vez instalado, podemos abrir un explorador web y escribir la dirección http://localhost:8080. Si se nos abre la página principal de Tomcat es que la instalación ha sido un éxito.

Cuando ya se dispone de todos los elementos necesarios para trabajar, se puede explicar la estructura que debe seguir la aplicación web dentro de los directorios de la instalación de Tomcat.

Se accede a los ficheros generados por parte de la instalación de Tomcat, generalmente en Archivos de Programa salvo que se le haya indicado otra ruta. Las aplicaciones web deben encontrarse en la siguiente ruta:

$CATALINA_HOME/webapps. En esa última carpeta, webapps, es donde se crearán cada una de las aplicaciones que tenga corriendo el servidor, por tanto, para nuestro proyecto fin de carrera creamos una nueva carpeta dentro de webapps llamada PFC. Por tanto, la ruta a nuestra aplicación será:

$CATALINA_HOME/webapps/PFC.

Una vez dentro de esta carpeta debemos crear una nueva carpeta, llamada WEB-INF, que más adelante se explicará para qué sirve. En este mismo directorio, es decir, $CATALINA_HOME/webapps/PFC es donde debemos incluir nuestras páginas JSP y todas las imágenes que disponga nuestra aplicación, al igual que las hojas de estilo. Ahora accedemos a la carpeta creada WEB-INF. Dentro de esta carpeta es donde irán alojadas las clases Java necesarias de nuestra aplicación web. Por ello creamos dos nuevas carpetas, una llamada classes y otro src. En la carpeta classes debemos incluir las clases compiladas, es decir, los .class, y en src, que no es requisito indispensable pero puede permanecer ahí, las clases con el código fuente.

En nuestro caso, necesitamos una librería externa para conectar con la base de datos, por tanto, en este mismo directorio, creamos una carpeta llamada lib, y dentro de ésta se aloja el archivo .jar correspondiente al conector JDBC para las bases de datos. Por último y para terminar, en este directorio $CATALINA_HOME/webapps/PFC /WEB-INF/ debemos incluir el archivo descriptor de la aplicación, llamado web.xml, donde se le indican el nombre de los Servlets y sus direcciones relativas para acceder desde la URL. En el siguiente Anexo de ficheros de configuración, se podrá observar dicho fichero creado para la aplicación web del proyecto fin de carrera.

Para terminar, hay que recordar que la compilación de los Servlets utiliza una librería externa a la que incorpora el JDK. Esta librería es servlets-api.jar, y se encuentra en la instalación de Tomcat en la carpeta lib. Lo que se puede hacer es incluirla en las variables de entorno del sistema en la variable CLASSPATH como ya se ha explicado anteriormente o bien utilizar la siguiente sentencia a la hora de compilar:

javac –classpath $CATALINA_HOME/lib/servlets-api.jar *.java

Sin duda la mejor opción, más cómoda y la que se recomienda es la primera.

Anexo F. Ficheros de configuración

1.- Servidor DNS primario

1.1.- named.conf

```
// This is the primary configuration file for the BIND DNS server named.
//
// Please read /usr/share/doc/bind9/README.Debian.gz for information on the
// structure of BIND configuration files in Debian, *BEFORE* you customize
// this configuration file.
//
// If you are just adding zones, please do that in /etc/bind/named.conf.local

include "/etc/bind/named.conf.options";

// prime the server with knowledge of the root servers
zone "." {
	type hint;
	file "/etc/bind/db.root";
};

// be authoritative for the localhost forward and reverse zones, and for
// broadcast zones as per RFC 1912

zone "localhost" {
	type master;
	file "/etc/bind/db.local";
};

zone "127.in-addr.arpa" {
	type master;
	file "/etc/bind/db.127";
};

zone "0.in-addr.arpa" {
	type master;
	file "/etc/bind/db.0";
};

zone "255.in-addr.arpa" {
	type master;
	file "/etc/bind/db.255";
};



zone "redSIP.fic" {
```

```
        type master;
        file "/etc/bind/db.redSIP";
        allow-transfer {
                192.168.240.105;
        };
};

include "/etc/bind/named.conf.local";
```

1.2.- db.redSIP

```
$TTL 86400
@       IN      SOA servidorDNS.redSIP.fic elias.redSIP.fic (
                        200909031 ; Serial formato: yyyymmddn donde n es un número cualquiera
                        10800 ; Refresh después de tres horas
                        3600 ; Reintentar después de una hora
                        604800 ; Expirar después de una semana
                        86400 ) ; TTL(Time to Live) mínimo de un día

@ IN ns servidorDNS.redSIP.fic.
redSIP.fic. IN A 192.168.240.100

servidorWEB IN A 192.168.240.2
servidorDNS IN A 192.168.240.100

pbxA IN A 192.168.240.115
pbxB IN A 192.168.240.120

www IN CNAME servidorWEB

_sip._udp.redSIP.fic. 300 IN SRV 0 0 5060 pbxA.redSIP.fic.
_sip._udp.redSIP.fic. 300 IN SRV 0 0 5060 pbxB.redSIP.fic.
```

1.3.- db.root

```
;       This file holds the information on root name servers needed to
;       initialize cache of Internet domain name servers
;       (e.g. reference this file in the "cache  .  <file>"
;       configuration file of BIND domain name servers).
;
;       This file is made available by InterNIC
;       under anonymous FTP as
;           file                /domain/named.root
;           on server           FTP.INTERNIC.NET
;       -OR-                    RS.INTERNIC.NET
;
```

```
;       last update:    Feb 04, 2008
;       related version of root zone:   2008020400
;
; formerly NS.INTERNIC.NET
;
.                       3600000  IN  NS    A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET.     3600000      A     198.41.0.4
A.ROOT-SERVERS.NET.     3600000      AAAA  2001:503:BA3E::2:30
;
; formerly NS1.ISI.EDU
;
.                       3600000      NS    B.ROOT-SERVERS.NET.
B.ROOT-SERVERS.NET.     3600000      A     192.228.79.201
;
; formerly C.PSI.NET
;
.                       3600000      NS    C.ROOT-SERVERS.NET.
C.ROOT-SERVERS.NET.     3600000      A     192.33.4.12
;
; formerly TERP.UMD.EDU
;
.                       3600000      NS    D.ROOT-SERVERS.NET.
D.ROOT-SERVERS.NET.     3600000      A     128.8.10.90
;
; formerly NS.NASA.GOV
;
.                       3600000      NS    E.ROOT-SERVERS.NET.
E.ROOT-SERVERS.NET.     3600000      A     192.203.230.10
;
; formerly NS.ISC.ORG
;
.                       3600000      NS    F.ROOT-SERVERS.NET.
F.ROOT-SERVERS.NET.     3600000      A     192.5.5.241
F.ROOT-SERVERS.NET.     3600000      AAAA  2001:500:2f::f
;
; formerly NS.NIC.DDN.MIL
;
.                       3600000      NS    G.ROOT-SERVERS.NET.
G.ROOT-SERVERS.NET.     3600000      A     192.112.36.4
;
; formerly AOS.ARL.ARMY.MIL
;
.                       3600000      NS    H.ROOT-SERVERS.NET.
H.ROOT-SERVERS.NET.     3600000      A     128.63.2.53
H.ROOT-SERVERS.NET.     3600000      AAAA  2001:500:1::803f:235
;
; formerly NIC.NORDU.NET
;
.                       3600000      NS    I.ROOT-SERVERS.NET.
I.ROOT-SERVERS.NET.     3600000      A     192.36.148.17
```

```
;
; operated by VeriSign, Inc.
;
.                       3600000      NS    J.ROOT-SERVERS.NET.
J.ROOT-SERVERS.NET.     3600000      A     192.58.128.30
J.ROOT-SERVERS.NET.     3600000      AAAA  2001:503:C27::2:30
;
; operated by RIPE NCC
;
.                       3600000      NS    K.ROOT-SERVERS.NET.
K.ROOT-SERVERS.NET.     3600000      A     193.0.14.129
K.ROOT-SERVERS.NET.     3600000      AAAA  2001:7fd::1
;
; operated by ICANN
;
.                       3600000      NS    L.ROOT-SERVERS.NET.
L.ROOT-SERVERS.NET.     3600000      A     199.7.83.42
;
; operated by WIDE
;
.                       3600000      NS    M.ROOT-SERVERS.NET.
M.ROOT-SERVERS.NET.     3600000      A     202.12.27.33
M.ROOT-SERVERS.NET.     3600000      AAAA  2001:dc3::35
; End of File
```

2.- Servidor DNS secundario

Sólo se muestra el fichero named.conf, ya que el fichero db.root es exactamente igual que el del servidor primario.

2.1.- named.conf

```
// This is the primary configuration file for the BIND DNS server named.
//
// Please read /usr/share/doc/bind9/README.Debian.gz for information on the
// structure of BIND configuration files in Debian, *BEFORE* you customize
// this configuration file.
//
// If you are just adding zones, please do that in /etc/bind/named.conf.local

include "/etc/bind/named.conf.options";

// prime the server with knowledge of the root servers
zone "." {
        type hint;
        file "/etc/bind/db.root";
};
```

```
// be authoritative for the localhost forward and reverse zones, and for
// broadcast zones as per RFC 1912

zone "localhost" {
	type master;
	file "/etc/bind/db.local";
};

zone "127.in-addr.arpa" {
	type master;
	file "/etc/bind/db.127";
};

zone "0.in-addr.arpa" {
	type master;
	file "/etc/bind/db.0";
};

zone "255.in-addr.arpa" {
	type master;
	file "/etc/bind/db.255";
};

zone “redSIP.fic” {
	type slave;
	file “/etc/bind/db
	master {
		192.168.240.100
	};
};

include "/etc/bind/named.conf.local
```

3.- IP-PBX Asterisk

A continuación se muestran los ficheros de configuración utilizados en las centralitas. La configuración es exactamente igual para todas las centralitas, excepto el fichero dundi.conf. Como se explicó en su correspondiente apartado, hay que modificar direcciones MAC e IP según la centralita que se esté configurando. Aquí sólo se reflejará la configuración para una de ellas.

3.1.- Archivo res_mysql.conf

```
;Sample configuration for res_config_mysql.c
;
; The value of dbhost may be either a hostname or an IP address.
; If dbhost is commented out or the string "localhost", a connection
```

```
; to the local host is assumed and dbsock is used instead of TCP/IP
; to connect to the server.
;
; Multiple database contexts may be configured, with the caveat that
; all context names should be unique and must not contain the slash ('/')
; character.  If you wish to separate reads from writes in your database
; configuration, you specify the database (NOT HERE, in other files)
; separated by a slash, read database first.  If your database
; specification does not contain a slash, the implication is that reads
; and writes should be performed to the same database.
;
; For example, in extconfig.conf, you could specify a line like:
;    sippeers => mysql,readhost.asterisk/writehost.asterisk,sipfriends
; and then define the contexts [readhost.asterisk] and [writehost.asterisk]
; below.
;
; The requirements parameter is available only in Asterisk 1.6.1 and
; later and must be present in all contexts.  It specifies the behavior
; when a column name is required by the system.  The default behavior is
; "warn" and simply sends a warning to the logger that the column does
; not exist (or is of the wrong type or precision).  The other two
; possibilities are "createclose", which adds the column with the right
; type and length, and "createchar", which adds the column as a char
; type, with the appropriate length to accept the data.  Note that with
; the MySQL driver, both "createclose" and "createchar" will, on occasion,
; widen a table column width to meet the requirements specified.
;
[general]
dbhost = localhost
dbname = asterisk
dbuser = userAsterisk
dbpass = 030704
dbport = 3306
dbsock = /var/run/mysqld/mysqld.sock
```

3.2.- Archivo extconfig.conf

```
;
; Static and realtime external configuration
; engine configuration
;
; Please read doc/extconfig.txt for basic table
; formatting information.
;
[settings]
;
; Static configuration files:
;
; file.conf => driver,database[,table]
;
```

```
; maps a particular configuration file to the given
; database driver, database and table (or uses the
; name of the file as the table if not specified)
;
;uncomment to load queues.conf via the odbc engine.
;
;queues.conf => odbc,asterisk,ast_config
;extensions.conf => sqlite,asterisk,ast_config
;
; The following files CANNOT be loaded from Realtime storage:
;       asterisk.conf
;       extconfig.conf (this file)
;       logger.conf
;
; Additionally, the following files cannot be loaded from
; Realtime storage unless the storage driver is loaded
; early using 'preload' statements in modules.conf:
;       manager.conf
;       cdr.conf
;       rtp.conf
;
;
; Realtime configuration engine
;
; maps a particular family of realtime
; configuration to a given database driver,
; database and table (or uses the name of
; the family if the table is not specified
;
;example => odbc,asterisk,alttable
;example2 => ldap,"dc=oxymium,dc=net",example2
;
; "odbc" is shown in the examples below, but is not the only valid realtime
; engine.  There is:
;    odbc ... res_config_odbc
;    sqlite ... res_config_sqlite
;    pgsql ... res_config_pgsql
;
;iaxusers => odbc,asterisk
;iaxpeers => odbc,asterisk
sipusers => mysql,general,usuariosSIP
sippeers => mysql,general,usuariosSIP
;sipregs => odbc,asterisk
;voicemail => odbc,asterisk
extensions => mysql,general,dialplan
;meetme => mysql,conferences
;queues => odbc,asterisk
;queue_members => odbc,asterisk
;musiconhold => mysql,asterisk
;queue_log => mysql,aasterisk
```

3.3.- Archivo dundi.conf

```
;
; DUNDi configuration file
;
; For more information about DUNDi, see http://www.dundi.com
;
[general]
;
; The "general" section contains general parameters relating
; to the operation of the dundi client and server.
;
; The first part should be your complete contact information
; should someone else in your peer group need to contact you.
;
department=Servidor A
organization=Ferreras, S.A.
locality=Pinto
stateprov=Madrid
country=Spain
email=your@email.com
phone=+12565551212
;
;
; Specify bind address and port number.  Default is
; 4520
;
bindaddr=0.0.0.0
port=4520
;
; See qos.tex or Quality of Service section of asterisk.pdf for a description of the tos parameter.
;tos=ef
;
; Our entity identifier (Should generally be the MAC address of the
; machine it's running on.  Defaults to the first eth address, but you
; can override it here, as long as you set it to the MAC of *something*
; you own!)  The EID can be overridden by a setting in asterisk.conf,
; or by setting this option.
;
entityid=00:0c:29:a4:1e:30
;
; Peers shall cache our query responses for the specified time,
; given in seconds. Default is 3600.
;
cachetime=10
;
; This defines the max depth in which to search the DUNDi system.
; Note that the maximum time that we will wait for a response is
; (2000 + 200 * ttl) ms.
;
```

```
ttl=32
;
; If we don't get ACK to our DPDISCOVER within 2000ms, and autokill is set
; to yes, then we cancel the whole thing (that's enough time for one
; retransmission only).  This is used to keep things from stalling for a long
; time for a host that is not available, but would be ill advised for bad
; connections.  In addition to 'yes' or 'no' you can also specify a number
; of milliseconds.  See 'qualify' for individual peers to turn on for just
; a specific peer.
;
autokill=yes
;
; pbx_dundi creates a rotating key called "secret", under the family
; 'secretpath'.  The default family is dundi (resulting in
; the key being held at dundi/secret).
;
;secretpath=dundi
;
; The 'storehistory' option (also changeable at runtime with
; 'dundi store history' and 'dundi no store history') will
; cause the DUNDi engine to keep track of the last several
; queries and the amount of time each query took to execute
; for the purpose of tracking slow nodes.  This option is
; off by default due to performance impacts.
;
;storehistory=yes

;[mappings]
;
; The "mappings" section maps DUNDi contexts
; to contexts on the local asterisk system.  Remember
; that numbers that are made available under the e164
; DUNDi context are regulated by the DUNDi General Peering
; Agreement (GPA) if you are a member of the DUNDi E.164
; Peering System.
;
; dundi_context => local_context,weight,tech,dest[,options]]
;
; 'dundi_context' is the name of the context being requested
; within the DUNDi request
;
; 'local_context' is the name of the context on the local system
; in which numbers can be looked up for which responses shall be given.
;
; 'weight' is the weight to use for the responses provided from this
; mapping.  The number must be >= 0 and < 60000.  Since it is totally
; valid to receive multiple responses to a query, responses received
; with a lower weight are tried first.  Note that the weight has a
; special meaning in the e164 context - see the GPA for more details.
;
```

; 'tech' is the technology to use (IAX, SIP, H323)
;
; 'dest' is the destination to supply for reaching that number. The
; following variables can be used in the destination string and will
; be automatically substituted:
; ${NUMBER}: The number being requested
; ${IPADDR}: The IP address to connect to
; ${SECRET}: The current rotating secret key to be used
;
[mappings]
troncalIAX=>registrosSIP,0,IAX2,dundi:${SECRET}@192.168.240.115/${NUMBER},nopartial

[00:0c:29:fe:c0:7c]
model=symmetric
host=192.168.240.120
inkey=claveDUNDi
outkey= claveDUNDi
include=troncalIAX
permit=troncalIAX
qualify=yes
order=primary

3.4- Archivo iax.conf

En el siguiente archivo lo único que nos interesa es la creación del peer para el canal IAX2 entre las centralitas, por lo que se obviará la etiqueta [general] ya que no se ha configurado nada.

; Inter-Asterisk eXchange driver definition
;
; This configuration is re-read at reload
; or with the CLI command
;   reload chan_iax2.so
;
; General settings, like port number to bind to, and
; an option address (the default is to bind to all
; local addresses).
;
[general]
.
.
.
[dundi]
type=user
context=llamada_DUNDi
dbsecret=dundi/secret
disallow=all
allow=ulaw

```
allow=alaw
allow=gsm
```

3.5.- Archivo sip.conf

El siguiente archivo tiene bastantes parámetros a configurar ya que configura todo lo relacionado con el protocolo SIP. Nuevamente sólo se mostrarán los apartados que realmente nos interesan.

```
;
; SIP Configuration example for Asterisk
;
; SIP dial strings
;-----------------------------------------------------------
; In the dialplan (extensions.conf) you can use several
; syntaxes for dialing SIP devices.
;       SIP/devicename
;       SIP/username@domain   (SIP uri)
;       SIP/username[:password[:md5secret[:authname[:transport]]]]@host[:port]
;       SIP/devicename/extension
;
;
; Devicename
;       devicename is defined as a peer in a section below.
;
; username@domain
;       Call any SIP user on the Internet
;       (Don't forget to enable DNS SRV records if you want to use this)
;
; devicename/extension
;       If you define a SIP proxy as a peer below, you may call
;       SIP/proxyhostname/user or SIP/user@proxyhostname
;       where the proxyhostname is defined in a section below
;       This syntax also works with ATA's with FXO ports
;
; SIP/username[:password[:md5secret[:authname]]]@host[:port]
;       This form allows you to specify password or md5secret and authname
;       without altering any authentication data in config.
;       Examples:
;
;       SIP/*98@mysipproxy
;       SIP/sales:topsecret::account02@domain.com:5062
;       SIP/12345678::bc53f0ba8ceb1ded2b70e05c3f91de4f:myname@192.168.0.1
;
; All of these dial strings specify the SIP request URI.
; In addition, you can specify a specific To: header by adding an
; exclamation mark after the dial string, like
;
;        SIP/sales@mysipproxy!sales@edvina.net
```

```
;
; CLI Commands
; ----------------------------------------------------------
; Useful CLI commands to check peers/users:
;   sip show peers              Show all SIP peers (including friends)
;   sip show users              Show all SIP users (including friends)
;   sip show registry           Show status of hosts we register with
;
;   sip set debug on            Show all SIP messages
;
;   module reload chan_sip.so    Reload configuration file
;                           Active SIP peers will not be reconfigured
;
;------- Naming devices -----------------------------------------------------
;
; When naming devices, make sure you understand how Asterisk matches calls
; that come in.
;       1. Asterisk checks the SIP From: address username and matches against
;          names of devices with type=user
;          The name is the text between square brackets [name]
;       2. Asterisk checks the From: addres and matches the list of devices
;          with a type=peer
;       3. Asterisk checks the IP address (and port number) that the INVITE
;          was sent from and matches against any devices with type=peer
;
; Don't mix extensions with the names of the devices. Devices need a unique
; name. The device name is *not* used as phone numbers. Phone numbers are
; anything you declare as an extension in the dialplan (extensions.conf).
;
; When setting up trunks, make sure there's no risk that any From: username
; (caller ID) will match any of your device names, because then Asterisk
; might match the wrong device.
;
; Note: The parameter "username" is not the username and in most cases is
;       not needed at all. Check below. In later releases, it's renamed
;       to "defaultuser" which is a better name, since it is used in
;       combination with the "defaultip" setting.
;-----------------------------------------------------------------------------

; ** Deprecated configuration options **
; The "call-limit" configuation option is deprecated. It still works in
; this version of Asterisk, but will disappear in the next version.
; You are encouraged to use the dialplan groupcount functionality
; to enforce call limits instead of using this channel-specific method.
;
; You can still set limits per device in sip.conf or in a database by using
; "setvar" to set variables that can be used in the dialplan for various limits.
```

```
[general]
context=exten_SIP            ; Default context for incoming calls
;allowguest=no               ; Allow or reject guest calls (default is yes)
;match_auth_username=yes     ; if available, match user entry using the
                             ; 'username' field from the authentication line
                             ; instead of the From: field.
allowoverlap=no              ; Disable overlap dialing support. (Default is yes)
;allowtransfer=no            ; Disable all transfers (unless enabled in peers or users)
                             ; Default is enabled
;realm=mydomain.tld          ; Realm for digest authentication
                             ; defaults to "asterisk". If you set a system name in
                             ; asterisk.conf, it defaults to that system name
                             ; Realms MUST be globally unique according to RFC  3261
                             ; Set this to your host name or domain name
udpbindaddr=0.0.0.0          ; IP address to bind UDP listen socket to (0.0.0.0 binds to all)
                             ; Optionally add a port number, 192.168.1.1:5062 (defaul  is port 5060)
srvlookup=yes                ; Enable DNS SRV lookups on outbound calls
                             ; Note: Asterisk only uses the first host
                             ; in SRV records
                             ; Disabling DNS SRV lookups disables the
                             ; ability to place SIP calls based on domain
                             ; names to some other SIP users on the Internet



;
; If regcontext is specified, Asterisk will dynamically create and destroy a
; NoOp priority 1 extension for a given peer who registers or unregisters with
; us and have a "regexten=" configuration item.
; Multiple contexts may be specified by separating them with '&'. The
; actual extension is the 'regexten' parameter of the registering peer or its
; name if 'regexten' is not provided.  If more than one context is provided,
; the context must be specified within regexten by appending the desired
; context after '@'.  More than one regexten may be supplied if they are
; separated by '&'.  Patterns may be used in regexten.
;
regcontext=sipregistrations
;regextenonqualify=yes        ; Default "no"
                              ;If you have qualify on and the peer becomes unreachable
                               ; this setting will enforce inactivation of the regexten
                               ; extension for the peer

;----------------------------------------- REALTIME SUPPORT -----------------------
; For additional information on ARA, the Asterisk Realtime Architecture,
; please read realtime.txt and extconfig.txt in the /doc directory of the
; source code.
;
rtcachefriends=yes           ; Cache realtime friends by adding them to the internal list
                             ; just like friends added from the config file only on a
                             ; as-needed basis? (yes|no)
```

```
;rtsavesysname=yes          ; Save systemname in realtime database at registration
                            ; Default= no

rtupdate=yes                ; Send registry updates to database using realtime? (yes|no)
                            ; If set to yes, when a SIP UA registers successfully, the ip address,
                          ; the origination port, the registration period, and the username of
                          ; the UA will be set to database via realtime.
                          ; If not present, defaults to 'yes'. Note: realtime peers will
                          ; probably not function across reloads in the way that you expect, if
                          ; you turn this option off.
;rtautoclear=yes           ; Auto-Expire friends created on the fly on the same schedule
                            ; as if it had just registered? (yes|no|<seconds>)
                            ; If set to yes, when the registration expires, the friend will
                            ; vanish from the configuration until requested again. If set
                            ; to an integer, friends expire within this number of seconds
                            ; instead of the registration interval.

;ignoreregexpire=yes        ; Enabling this setting has two functions:
                            ;
                           ; For non-realtime peers, when their registration expires, the
                           ; information will _not_ be removed from memory or the Asterisk database
                          ; if you attempt to place a call to the peer, the existing information
                          ; will be used in spite of it having expired
                          ;
                          ; For realtime peers, when the peer is retrieved from realtime storage,
                          ; the registration information will be used regardless of whether
                          ; it has expired or not; if it expires while the realtime peer
                          ; is still in memory (due to caching or other reasons), the
                          ; information will not be removed from realtime storage
```

3.6.- Archivo extensions.conf

```
; extensions.conf - the Asterisk dial plan
;
; Static extension configuration file, used by
; the pbx_config module. This is where you configure all your
; inbound and outbound calls in Asterisk.
;
; This configuration file is reloaded
; - With the "dialplan reload" command in the CLI
; - With the "reload" command (that reloads everything) in the CLI

;
; The "General" category is for certain variables.
;
[general]
;
; If static is set to no, or omitted, then the pbx_config will rewrite
; this file when extensions are modified.  Remember that all comments
; made in the file will be lost when that happens.
```

```
;
; XXX Not yet implemented XXX
;
static=yes
;
; if static=yes and writeprotect=no, you can save dialplan by
; CLI command "dialplan save" too
;
writeprotect=no
;
; If autofallthrough is set, then if an extension runs out of
; things to do, it will terminate the call with BUSY, CONGESTION
; or HANGUP depending on Asterisk's best guess. This is the default.
;
; If autofallthrough is not set, then if an extension runs out of
; things to do, Asterisk will wait for a new extension to be dialed
; (this is the original behavior of Asterisk 1.0 and earlier).
;
;autofallthrough=no
;
; NOTE: A complication sets in, if you put your global variables into
; the AEL file, instead of the extensions.conf file. With clearglobalvars
; set, a "reload" will often leave the globals vars cleared, because it
; is not unusual to have extensions.conf (which will have no globals)
; load after the extensions.ael file (where the global vars are stored).
; So, with "reload" in this particular situation, first the AEL file will
; clear and then set all the global vars, then, later, when the extensions.conf
; file is loaded, the global vars are all cleared, and then not set, because
; they are not stored in the extensions.conf file.
;
clearglobalvars=no
;
; WARNING WARNING WARNING WARNING
; If you load any other extension configuration engine, such as pbx_ael.so,
; your global variables may be overridden by that file.  Please take care to
; use only one location to set global variables, and you will likely save
; yourself a ton of grief.
; WARNING WARNING WARNING WARNING
;
; Any category other than "General" and "Globals" represent
; extension contexts, which are collections of extensions.
;
; Extension names may be numbers, letters, or combinations
; thereof. If an extension name is prefixed by a '_'
; character, it is interpreted as a pattern rather than a
; literal.  In patterns, some characters have special meanings:
;
;   X - any digit from 0-9
;   Z - any digit from 1-9
;   N - any digit from 2-9
```

```
;   [1235-9] - any digit in the brackets (in this example, 1,2,3,5,6,7,8,9)
;   . - wildcard, matches anything remaining (e.g. _9011. matches
;       anything starting with 9011 excluding 9011 itself)
;   ! - wildcard, causes the matching process to complete as soon as
;       it can unambiguously determine that no other matches are possible
;
; For example, the extension _NXXXXXX would match normal 7 digit dialings,
; while _1NXXNXXXXXX would represent an area code plus phone number
; preceded by a one.
;
; Each step of an extension is ordered by priority, which must always start
; with 1 to be considered a valid extension.  The priority "next" or "n" means
; the previous priority plus one, regardless of whether the previous priority
; was associated with the current extension or not.  The priority "same" or "s"
; means the same as the previously specified priority, again regardless of
; whether the previous entry was for the same extension.  Priorities may be
; immediately followed by a plus sign and another integer to add that amount
; (most useful with 's' or 'n').  Priorities may then also have an alias, or
; label, in parentheses after their name which can be used in goto situations.
;
; Contexts contain several lines, one for each step of each extension.  One may
; include another context in the current one as well, optionally with a date
; and time.  Included contexts are included in the order they are listed.
; Switches may also be included within a context.  The order of matching within
; a context is always exact extensions, pattern match extensions, includes, and
; switches.  Includes are always processed depth-first.  So for example, if you
; would like a switch "A" to match before context "B", simply put switch "A" in
; an included context "C", where "C" is included in your original context
; before "B".
;
[llamada_DUNDi]
exten => _2XX,1,Goto(exten_SIP,${EXTEN},1)

[internal]
include => busqueda_DUNDi
include => exten_SIP

[busqueda_DUNDi]
switch => DUNDi/troncalIAX

[exten_SIP]
switch => Realtime/dialplan
```

3.7 Fichero cdr_mysql.conf

```
; Note - if the database server is hosted on the same machine as the
; asterisk server, you can achieve a local Unix socket connection by
; setting hostname=localhost
;
```

```
; port and sock are both optional parameters.  If hostname is specified
; and is not "localhost" (you can use address 127.0.0.1 instead), then
; cdr_mysql will attempt to connect to the port specified or use the
; default port.  If hostname is not specified or if hostname is
; "localhost", then cdr_mysql will attempt to connect to the socket file
; specified by sock or otherwise use the default socket file.
;
[global]
hostname=localhost
dbname=asterisk
table=cdr
password=030704
user=userAsterisk
port=3306
sock=/var/run/mysqld/mysqld.sock
```

4.- Servidor web. Archivo descriptor de la aplicación

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<web-app xmlns=http://java.sun.com/xml/ns/javaee
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  version="2.5">

 <display-name>Sistema de supervision</display-name>
 <description>
   Aplicacion web para monitorizar las centralitas Asterisk
 </description>

 <servlet>
        <servlet-name>direcciones</servlet-name>
        <servlet-class>ServletObtencionIP</servlet-class>
 </servlet>

 <servlet-mapping>
  <servlet-name>direcciones</servlet-name>
  <url-pattern>/direcciones</url-pattern>
 </servlet-mapping>


 <servlet>
        <servlet-name>nuevaExtension</servlet-name>
        <servlet-class>ServletNuevaExtension</servlet-class>
 </servlet>
```

```
<servlet-mapping>
  <servlet-name>nuevaExtension</servlet-name>
  <url-pattern>/nuevaExtension</url-pattern>
</servlet-mapping>

<servlet>
        <servlet-name>recargar</servlet-name>
        <servlet-class>ServletRecargaInicio</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>recargar</servlet-name>
  <url-pattern>/recargar</url-pattern>
</servlet-mapping>

<servlet>
        <servlet-name>nuevaEntradaDP</servlet-name>
        <servlet-class>ServletNuevaDP</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>nuevaEntradaDP</servlet-name>
  <url-pattern>/nuevaEntradaDP</url-pattern>
</servlet-mapping>

<servlet>
        <servlet-name>verExtensiones</servlet-name>
        <servlet-class>ServletVerUsuarios</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>verExtensiones</servlet-name>
  <url-pattern>/verExtensiones</url-pattern>
</servlet-mapping>

<servlet>
        <servlet-name>resetearBBDD</servlet-name>
        <servlet-class>ServletReset</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>resetearBBDD</servlet-name>
  <url-pattern>/resetearBBDD</url-pattern>
</servlet-mapping>
<servlet>
        <servlet-name>desconexion</servlet-name>
        <servlet-class>ServletDesconexion</servlet-class>
</servlet>
```

```
  <servlet-mapping>
    <servlet-name>desconexion</servlet-name>
    <url-pattern>/desconexion</url-pattern>
  </servlet-mapping>

  <servlet>
        <servlet-name>comprobar</servlet-name>
        <servlet-class>ServletCompruebaDireccion</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>comprobar</servlet-name>
    <url-pattern>/comprobar</url-pattern>
  </servlet-mapping>

  <servlet>
        <servlet-name>verCDR</servlet-name>
        <servlet-class>ServletVerCDR</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>verCDR</servlet-name>
    <url-pattern>/verCDR</url-pattern>
  </servlet-mapping>

</web-app>
```

ANEXO G. API JAVADOC

Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS | FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD | DETAIL: FIELD | CONSTR | METHOD

Class Conexion

```
java.lang.Object
  Conexion
```

```
public class Conexion
extends java.lang.Object
```

Constructor Summary

`Conexion()`
Método constructor de la clase.

`Conexion(java.lang.String direccion)`
Método constructor de la clase.

Method Summary

void	`cerrarConexion()` Metodo para cerrar la conexión con la base de datos
boolean	`comprobarConexion(java.lang.String direccion)` Método que permite saber si la conexión con la base de datos es posible
boolean	`comprobarIP(java.lang.String ip)` Metodo que nos permite saber si la dirección IP introducida por el usuario tiene un formato correcto
boolean	`nuevaEntradaDialPlan(java.lang.String context, java.lang.String exten, java.lang.String priority, java.lang.String app, java.lang.String appdata)` Metodo que permite introducir nuevas sentencias de marcación para el Dialplan
boolean	`nuevaExtension(java.lang.String extension, java.lang.String secret, java.lang.String userName)` Metodo que permite introducir nuevas extensiones en la base de datos
boolean	`resetearUsuarios()` Metodo que permite dejar la tabla usuariosSIP con las direcciones IP y puertos reseteados a 0.0.0.0 y 0 respectivamente
java.util.Vector	`verRegistroLLamadas()` Metodo que permite obtener los datos necesarios de la tabla cdr para ver el registro de llamadas
java.util.Vector	`verUsuariosConectados()` Metodo que obtiene los usuarios registrados en la centralita que deseemos.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Conexion

```
public Conexion()
```

Método constructor de la clase. Está vacío porque servirá para utilizar sus métodos que no corresponden a una conexión directa con las bases de datos, como comprobarConexion() o comprobarIP(), en los Servlets

---

Conexion

```
public Conexion(java.lang.String direccion)
```

Método constructor de la clase. Crea una conexión con la base de datos que se pasa por parametro

Parameters:

`direccion` - Direccion IP de la base de datos con la que queremos establecer conexion

Throws:

`exception` - si existe algun problema con la conexión a la base datos. Estas excepciones podrán ser del tipo: "SQLException", "InstantiationException", "ClassNotFoundException" y si existe cualquier otro fallo "Exception", capturadas con sentencia try-catch

Method Detail

comprobarConexion

```
public boolean comprobarConexion(java.lang.String direccion)
```

Método que permite saber si la conexión con la base de datos es posible

Parameters:

`direccion` - Direccion IP de la base de datos de la cual queremos saber si es posible su conexion

Returns:

true o false dependiendo de la disponibilidad de la base de datos

Throws:

`exception` - Si existe algún problema con la conexión a la base datos. Estas excepciones podrán ser del tipo: "SQLException", "InstantiationException", "ClassNotFoundException" y si existe cualquier otro fallo "Exception", capturadas con sentencia try-catch

---

nuevaExtension

```
public boolean nuevaExtension(java.lang.String extension,
                              java.lang.String secret,
                              java.lang.String userName)
```

Metodo que permite introducir nuevas extensiones en la base de datos

Parameters:

`extension` - Numero de extension que queremos introducir
`secret` - Clave secreta para la nueva extension utilizada en los telefonos IP
`userName` - Nombre de usuario asignado a la nueva extension

Returns:

true o false dependiendo de si se puede realizar la insercion

Throws:

`java.lang.Exception` - Captura cualquier anomalía mediante sentencia try-catch

---

nuevaEntradaDialPlan

```
public boolean nuevaEntradaDialPlan(java.lang.String context,
                                    java.lang.String exten,
                                    java.lang.String priority,
                                    java.lang.String app,
                                    java.lang.String appdata)
```

Metodo que permite introducir nuevas sentencias de marcación para el Dialplan

Parameters:

`context` - Nombre del contexto de llamada
`exten` - Extension a la cual se le aplica esta nueva sentencia
`priority` - Prioridad de sentencia respecto al resto de esta misma extension
`app` - Aplicacion que se ejecuta (Dial, hangup)
`appdata` - Datos necesarios para ejecutar la aplicacion

Returns:

true o false dependiendo de si se puede realizar la insercion

Throws:

`java.lang.Exception` - Captura cualquier anomalía mediante sentencia try-catch

---

verUsuariosConectados

```
public java.util.Vector verUsuariosConectados()
```

Metodo que obtiene los usuarios registrados en la centralita que deseemos. Se mostrarán sólo los usuarios registrados, es decir, aquellos cuya dirección IP sea distinta de 0.0.0.0

Returns:

Vector con las filas de la tabla y sus correspondientes datos (name,ipaddr,port,regseconds)

Throws:

`java.lang.Exception` - Captura cualquier anomalía mediante sentencia try-catch

---

verRegistroLLamadas

```
public java.util.Vector verRegistroLLamadas()
```

Metodo que permite obtener los datos necesarios de la tabla cdr para ver el registro de llamadas

Returns:

Vector con las filas de la tabla y sus correspondientes datos (calldate,src,dst,dcontext...). Cada posicion del vector contiene otro vector que representa la fila de la tabla obtenida en la consulta

Throws:

`java.lang.Exception` - Captura cualquier anomalía mediante sentencia try-catch

---

resetearUsuarios

```
public boolean resetearUsuarios()
```

Metodo que permite dejar la tabla usuariosSIP con las direcciones IP y puertos reseteados a 0.0.0.0 y 0 respectivamente

Returns:

true o false dependiendo de si se puede realizar la actualizacion

Throws:

`java.lang.Exception` - Captura cualquier anomalía mediante sentencia try-catch

---

cerrarConexion

```
public void cerrarConexion()
```

Metodo para cerrar la conexión con la base de datos

Throws:

`java.lang.Exception` - Captura cualquier anomalía mediante sentencia try-catch

---

comprobarIP

```
public boolean comprobarIP(java.lang.String ip)
```

Metodo que nos permite saber si la dirección IP introducida por el usuario tiene un formato correcto

Parameters:

`ip` - Dirección IP que se quiere comprobar

Returns:

true o false dependiendo del resultado obtenido

Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class ServletObtencionIP

```
java.lang.Object
  javax.servlet.GenericServlet
     javax.servlet.http.HttpServlet
          ServletObtencionIP
```

All Implemented Interfaces:

java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

```
public class ServletObtencionIP
extends javax.servlet.http.HttpServlet
```

See Also:

Serialized Form

Constructor Summary

ServletObtencionIP()

Method Summary

protected void	añadirEstadoIP(java.lang.String dirIP) Método encargado de saber si las bases de datos estan disponibles.
void	doPost(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) Método encargado de recibir los parámetros del formulario rellenado por el usuario para conocer las IP's.

Methods inherited from class javax.servlet.http.HttpServlet

doDelete, doGet, doHead, doOptions, doPut, doTrace, getLastModified, service, service

Methods inherited from class javax.servlet.GenericServlet

destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, init, log, log

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

ServletObtencionIP

```
public ServletObtencionIP()
```

Method Detail

doPost

```
public void doPost(javax.servlet.http.HttpServletRequest request,
                   javax.servlet.http.HttpServletResponse response)
            throws java.io.IOException,
                   javax.servlet.ServletException
```

Método encargado de recibir los parámetros del formulario rellenado por el usuario para conocer las IP's. Enviará al jsp un vector con el estado de las bases de datos.

Overrides:

`doPost` in class `javax.servlet.http.HttpServlet`

Parameters:

`request` - Variable que nos permite recuperar las direcciones del formulario

`response` - Variable necesaria para la correcta construcción del método doPost()

Throws:

`IOException,ServletException`

`java.io.IOException`

`javax.servlet.ServletException`

---

añadirEstadoIP

```
protected void añadirEstadoIP(java.lang.String dirIP)
```

Método encargado de saber si las bases de datos estan disponibles. Para ello primero comprobará si las direcciones IP introducidas por el usuario tienen un formato correcto haciendo uso del método comprobarIP(String ip) de la clase Conexion. De ser asi, cuando la direccion IP sea correcta la guardará en la sesión y el estado de la base de datos en el vector que se pasará al jsp.

Parameters:

`dirIP` - Direccion IP introducida por el usuario

Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class ServletCompruebaDireccion

```
java.lang.Object
  javax.servlet.GenericServlet
      javax.servlet.http.HttpServlet
          ServletCompruebaDireccion
```

All Implemented Interfaces:

java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

```
public class ServletCompruebaDireccion
extends javax.servlet.http.HttpServlet
```

See Also:

Serialized Form

Constructor Summary

ServletCompruebaDireccion()

Method Summary

void	doGet(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) Método encargado de ejecutar el Servlet tras recibir la petición desde el jsp para posteriormente redirigir al usuario a la página jsp pertinente.

Methods inherited from class javax.servlet.http.HttpServlet

doDelete, doHead, doOptions, doPost, doPut, doTrace, getLastModified, service, service

Methods inherited from class javax.servlet.GenericServlet

destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, init, log, log

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

ServletCompruebaDireccion

```
public ServletCompruebaDireccion()
```

Method Detail

doGet

```
public void doGet(javax.servlet.http.HttpServletRequest request,
                  javax.servlet.http.HttpServletResponse response)
           throws java.io.IOException,
                  javax.servlet.ServletException
```

Método encargado de ejecutar el Servlet tras recibir la petición desde el jsp para posteriormente redirigir al usuario a la página jsp pertinente. Comprobará si hay más de una dirección IP correcta en la sesión.

Overrides:

`doGet` in class `javax.servlet.http.HttpServlet`

Parameters:

`request` - Variable que nos permite obtener las direcciones de la sesión y la pagina a la que quiere acceder el usuario.

`response` - Variable necesaria para la correcta construcción del método doGet()

Throws:

`IOException,ServletException`

`java.io.IOException`

`javax.servlet.ServletException`

Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class ServletNuevaExtension

```
java.lang.Object
  javax.servlet.GenericServlet
      javax.servlet.http.HttpServlet
          ServletNuevaExtension
```

All Implemented Interfaces:

java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

```
public class ServletNuevaExtension
extends javax.servlet.http.HttpServlet
```

See Also:

Serialized Form

Constructor Summary

ServletNuevaExtension()

Method Summary

void	doPost(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) Método encargado de recibir los parámetros del formulario rellenado por el usuario con los datos necesarios para la nueva extensión.

Methods inherited from class javax.servlet.http.HttpServlet

doDelete, doGet, doHead, doOptions, doPut, doTrace, getLastModified, service, service

Methods inherited from class javax.servlet.GenericServlet

destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, init, log, log

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

ServletNuevaExtension

```
public ServletNuevaExtension()
```

Method Detail

doPost

```
public void doPost(javax.servlet.http.HttpServletRequest request,
                   javax.servlet.http.HttpServletResponse response)
            throws java.io.IOException,
                   javax.servlet.ServletException
```

Método encargado de recibir los parámetros del formulario rellenado por el usuario con los datos necesarios para la nueva extensión. Realiza la inserción en la base de datos si es posible y redirige al usuario al jsp para mostrarle el resultado. Para la inserción, hará uso del método nuevaExtension(String exten,String secret, String username) de la clase Conexion.

Overrides:

`doPost` in class `javax.servlet.http.HttpServlet`

Parameters:

`request` - Variable que permite recuperar las direcciones de la sesión y los parámetros del formulario.

`response` - Variable necesaria para la correcta construcción del método doPost()

Throws:

`IOException,ServletException`

`java.io.IOException`

`javax.servlet.ServletException`

Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS | FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD | DETAIL: FIELD | CONSTR | METHOD

Class ServletVerUsuarios

```
java.lang.Object
  javax.servlet.GenericServlet
      javax.servlet.http.HttpServlet
          ServletVerUsuarios
```

All Implemented Interfaces:

java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

```
public class ServletVerUsuarios
extends javax.servlet.http.HttpServlet
```

See Also:

Serialized Form

Constructor Summary

ServletVerUsuarios()

Method Summary

void	doGet(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) Método encargado de iniciar la ejecución solicitada en el jsp, en este caso ver los usuarios registrados.

Methods inherited from class javax.servlet.http.HttpServlet

doDelete, doHead, doOptions, doPost, doPut, doTrace, getLastModified, service, service

Methods inherited from class javax.servlet.GenericServlet

destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, init, log, log

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

ServletVerUsuarios

```
public ServletVerUsuarios()
```

Method Detail

doGet

```
public void doGet(javax.servlet.http.HttpServletRequest request,
                  javax.servlet.http.HttpServletResponse response)
           throws java.io.IOException,
                  javax.servlet.ServletException
```

Método encargado de iniciar la ejecución solicitada en el jsp, en este caso ver los usuarios registrados. Para ello hará uso del método verUsuariosConectados() de la clase Conexion, entre otros. Devolverá al jsp correspondiente un Vector con los datos solicitados.

Overrides:

`doGet` in class `javax.servlet.http.HttpServlet`

Parameters:

`request` - Variable para recuperar las direcciones de la sesión

`response` - Variable necesaria para la correcta construcción del método doGet()

Throws:

`IOException,ServletException`

`java.io.IOException`

`javax.servlet.ServletException`

Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class ServletVerCDR

```
java.lang.Object
  javax.servlet.GenericServlet
      javax.servlet.http.HttpServlet
          ServletVerCDR
```

All Implemented Interfaces:

java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

```
public class ServletVerCDR
extends javax.servlet.http.HttpServlet
```

See Also:

Serialized Form

Constructor Summary

ServletVerCDR()

Method Summary

void	doGet(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) Metodo encargado de iniciar la ejecución solicitada en el jsp, en este caso ver el registro de llamadas.

Methods inherited from class javax.servlet.http.HttpServlet

doDelete, doHead, doOptions, doPost, doPut, doTrace, getLastModified, service, service

Methods inherited from class javax.servlet.GenericServlet

destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, init, log, log

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

ServletVerCDR

```
public ServletVerCDR()
```

Method Detail

doGet

```
public void doGet(javax.servlet.http.HttpServletRequest request,
                  javax.servlet.http.HttpServletResponse response)
           throws java.io.IOException,
                  javax.servlet.ServletException
```

Metodo encargado de iniciar la ejecución solicitada en el jsp, en este caso ver el registro de llamadas. Para ello hará uso del método verRegistroLLamadas() de la clase Conexion, entre otros. Devolverá al jsp correspondiente un Vector con los datos solicitados

Overrides:

`doGet` in class `javax.servlet.http.HttpServlet`

Parameters:

`request` - Variable para recuperar las direcciones de la sesión

`response` - Variable necesaria para la correcta construcción del método doGet()

Throws:

`IOException,ServletException`

`java.io.IOException`

`javax.servlet.ServletException`

Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class ServletReset

```
java.lang.Object
  javax.servlet.GenericServlet
      javax.servlet.http.HttpServlet
          ServletReset
```

All Implemented Interfaces:

java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

```
public class ServletReset
extends javax.servlet.http.HttpServlet
```

See Also:

Serialized Form

Constructor Summary

ServletReset()

Method Summary

void	doPost(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) Metodo encargado de recibir los parametros del formulario rellenado por el usuario para conocer la dirección o o direcciones de la base de datos que se desea resetear su tabla usuariosSIP.

Methods inherited from class javax.servlet.http.HttpServlet

doDelete, doGet, doHead, doOptions, doPut, doTrace, getLastModified, service, service

Methods inherited from class javax.servlet.GenericServlet

destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, init, log, log

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

ServletReset

```
public ServletReset()
```

Method Detail

doPost

```
public void doPost(javax.servlet.http.HttpServletRequest request,
                   javax.servlet.http.HttpServletResponse response)
            throws java.io.IOException,
                   javax.servlet.ServletException
```

Metodo encargado de recibir los parametros del formulario rellenado por el usuario para conocer la dirección o o direcciones de la base de datos que se desea resetear su tabla usuariosSIP. Hará el reseteo gracias al método resetearUsuarios() de la clase Conexion. Redirige al usuario al jsp que le indicará si ha sido posible la acción.

Overrides:

`doPost` in class `javax.servlet.http.HttpServlet`

Parameters:

`request` - Variable que nos permite recuperar los datos del formulario.

`response` - Variable necesaria para la correcta construcción del método doPost()

Throws:

`IOException,ServletException`

`java.io.IOException`

`javax.servlet.ServletException`

Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS | FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD | DETAIL: FIELD | CONSTR | METHOD

Class ServletDesconexion

```
java.lang.Object
  javax.servlet.GenericServlet
      javax.servlet.http.HttpServlet
          ServletDesconexion
```

All Implemented Interfaces:

java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

```
public class ServletDesconexion
extends javax.servlet.http.HttpServlet
```

See Also:

Serialized Form

Constructor Summary

ServletDesconexion()

Method Summary

void	doGet(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) Método encargado de anular la sesión y redirigir al usaurio al inicio de la aplicación

Methods inherited from class javax.servlet.http.HttpServlet

doDelete, doHead, doOptions, doPost, doPut, doTrace, getLastModified, service, service

Methods inherited from class javax.servlet.GenericServlet

destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, init, log, log

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

ServletDesconexion

```
public ServletDesconexion()
```

Method Detail

doGet

```
public void doGet(javax.servlet.http.HttpServletRequest request,
                  javax.servlet.http.HttpServletResponse response)
           throws java.io.IOException,
                  javax.servlet.ServletException
```

Método encargado de anular la sesión y redirigir al usaurio al inicio de la aplicación

Overrides:

`doGet` in class `javax.servlet.http.HttpServlet`

Parameters:

`request` - Variable para recuperar la sesión

`response` - Variable necesaria para la correcta construcción del método doGet()

Throws:

`IOException,ServletException`

`java.io.IOException`

`javax.servlet.ServletException`

ANEXO H. Código fuente de los JavaServer Pages (JSP)

A continuación se va a mostrar el código fuente de las páginas JSP generadas para el desarrollo d la aplicación. Sólo se mostrará en su totalidad la primera página, es decir, la de la portada, por una sencilla razón. Las páginas están maquetadas con sentencias div, haciendo uso de una hoja de estilo. Las páginas se dividen por tanto en encabezado, menú izquierdo, panel de datos y pie de página. De todas ellas, la única que cambia a lo largo de la aplicación es la del panel de datos, ya que es ahí donde se muestran los resultados de la ejecución de las diferentes opciones del sistema. Por esta razón, a partir de Portada.jsp, sólo se mostrará el contenido alojado en las sentencias div pertenecientes al panel de datos.

1.- Portada.jsp

Como su propio nombre indica, es la portada de la aplicación. Aquí será donde el usuario deberá introducir las direcciones IP con las que desea trabajar.

```
<?xml version='1.0' encoding='ISO-8859-1' ?>

<%@ page language="java" contentType="text/html" %>

<!DOCTYPE html
        PUBLIC "-//W3C//DTD XHTML 1.1//EN"
        "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd" >

<html xmlns='http://www.w3.org/1999/xhtml'>
  <head>
    <link href="estilo.css" rel="stylesheet" type="text/css"> </link>
    <title>Sistema de supervisi&oacute;n</title>
  </head>
  <body>

    <!-- ENCABEZADO -->
          <div class="encabezado">
                  <H1> <img src='Logo_asterisk.png'>  MONITOR DE CENTRALITAS
ASTERISK </H1>
                  <hr class="linea">
          </div>

          <!-- MENU DE LA IZQUIERDA -->
          <div class="menuIzda">
                  <table width="95%" align="center" border="1">
                          <tr> <td bgcolor="#FFA500"> <a href='Error.jsp'> Inicio </a> </td>
</tr>
                          <tr> <td bgcolor="#FFA500"> <a href='Error.jsp'> Recomendaciones
</a> </td>
                          <tr> <td bgcolor="#FFA500"> <a href='Error.jsp'> Crear nuevas
Extensiones </a> </td> </tr>
```

```
                        <tr> <td bgcolor="#FFA500"> <a href='Error.jsp'> Ver usuarios
conectados </a> </td> </tr>
                        <tr> <td bgcolor="#FFA500"> <a href='Error.jsp'> Crear nuevas
entradas para el Dialplan </a> </td> </tr>
                        <tr> <td bgcolor="#FFA500"> <a href='Error.jsp'> Ver registro de
llamadas de las centralitas </a> </td> </tr>
                        <tr> <td bgcolor="#FFA500"> <a href='Error.jsp'> Resetear BBDD
</a> </td> </tr>
                </table>
        </div>

        <!-- PANEL CENTRAL DE RESULTADOS -->
        <div class="panel">
                <p> Introduzca la direcci&oacute;n IP de la centralita Asterisk que desea
monitorizar.
                        Si dispone de varias centralitas, escriba sus direcciones separadas por
","</p>
                <p> Ej: 192.168.0.1,192.168.0.2 </p>

                <form action ="direcciones" method ="post" >

                        <label for="direcciones">Direccion/es IP :</label>
                        <input type ="text" name="direcciones" id="direcciones" />

                        <p><input class="submit" type="submit" value="Enviar" /></p>

                </form>

        </div>

        <!-- PIE DE PAGINA -->
        <div class="pie">
                <p> Proyecto Fin de Carrera: Red de alta disponibilidad basada en DUNDI y
DNS SRV con
                        Sistema de Supervisión. El&iacute;as Ferreras Sarmiento. Universidad
Carlos III
                        de Madrid Escuela Polit&eacute;cnica </p>
        </div>
  </body>
</html>
```

2.- PaginaPrincipal.jsp

Esta página corresponde con la inmediatamente posterior a la introducción de las direcciones IP en la portada, mostrando el estado de las bases de datos o si las direcciones no fueron bien escritas. También se accede a esta página cuando el administrador pulsa la opción Inicio del panel de la izquierda:

```
<!-- PANEL CENTRAL DE RESULTADOS -->
<div class="panel">
<p> Se disponen de las siguientes centralitas Asterisk para trabajar </p>
<% Vector estado = (Vector)request.getAttribute("estado");
Vector direcciones = (Vector)session.getAttribute("direcciones");
 int numCentralita = direcciones.size();
 int numIP = estado.size(); %>

<ul> <p>N&uacute;mero de Centralitas <%= numCentralita %> </p>
<% if (numIP==1) {
      if(((String)estado.elementAt(0)).equals("vacio")) { %>
      <li> No dispone de direcciones IP para trabajar. Le recomendamos que  salga de la aplicaci&oacute e introduzca las direcciones IP necesarias </li>
      <% } else {
         if(!((String)estado.elementAt(0)).equals("true")&&!((String)estado.elementAt(0)).equals("false")) { %>
         <li> La direcci&oacute;n IP "<%= (String)estado.elementAt(0) %>" no tiene un formato correcto </li>
          <% } else { %>
                <li> Centralita Asterisk con direcci&oacute;n IP: <%= (String)direcciones.elementAt(0) %>
         <ul>
         <% if (((String)estado.elementAt(0)).equals("true")) { %>
                <li> Su base de datos est&aacute;:  Activa <img src='correcto.png'>  </li>
         <% } else { %>
                <li> Su base de datos est&aacute;:  No disponible <img src='incorrecto.jpg'> </li>
               <% }
         }
         } %>

         </ul>
         </li>
         <% } else {
                int j=0;
                for (int i=0; i<numIP; i++) { %>
                      <p></p>
                      <%if(!((String)estado.elementAt(i)).equals("true")&&
!((String)estado.elementAt(i)).equals("false")) { %>
                      <li> La direcci&oacute;n IP "<%= (String)estado.elementAt(i) %>" no tiene un formato correcto </li>
                      <% } else { %>
                            <li> Centralita Asterisk con direcci&oacute;n IP: <%= (String)direcciones.elementAt(j) %>
         <ul>
                <% if (((String)estado.elementAt(i)).equals("true")) { %>
                <li> Su base de datos est&aacute;:  Activa <img src='correcto.png'> </li>
```

```
        <% } else { %>
                    <li> Su base de datos est&aacute;:  No disponible <img src='incorrecto.jpg'>
</li>
                  <% }
        j++; %>
        </ul>
        </li>
        <%      }
         }
         } %>
</ul>
<br>
<p> TOPOLOG&Iacute;A DE RED: </p>

<% switch(numCentralita) {
        case 1: %> <img src='red1.jpg'> <% break;
        case 2: %> <img src='red2.jpg'> <% break;
        case 3: %> <img src='red3.jpg'> <% break;
} %>

<p> IMPORTANTE: Compruebe que las direcciones IP de las centralitas Asterisk sea correcta. De no ser así, vaya a la opci&oacute;n Salir del panel de la izquierda e introduzca nuevamente las direcciones. Si el problema persiste, contacte con el administrador del sistema. </p> </div>
</br>
</div>
```

3.- Recomendaciones.jsp

Página encargada de mostrar ciertas pautas al administrador de red con el fin del correcto funcionamiento del sistema se supervisión:

```
<!-- PANEL CENTRAL DE RESULTADOS -->
<div class="panel">
<p> A continuaci&oacute;n se pretende dar una serie de pautas para el correcto funcionamiento tanto del sistema de supervisi&oacute;n como del cl&uacute;ster de centralitas Asterisk. Es importante seguir estas pautas ya que las centralitas podr&iacute;an sufrir comportamientos no deseados: </p>

<ul>
<p> <li> No es necesario ejecutar el sistema de supervisi&oacute;n con todas las direcciones de las centralitas que forman el cl&uacute;ster ya que se pueden monitorizar de forma independiente. S&oacute;lo ser&aacute; obligatorio el uso de todas las direcciones cuando se quiera efectuar cambios en las bases de datos, del estilo "Crear nuevas extensiones" o "Crear nuevas entradas para el Dialplan". </li> </p>

<p> <li> Si al introducir las direcciones IP el sistema le advierte que al menos una de ellas no tiene el formato correcto, es aconsejable que salga del sistema y vuelva a introducir las direcciones. Esto se debe a que la aplicaci&oacute;n sólo trabajará con direcciones correctas. </li> </p>

<p> <li> Siempre que se introduzca una nueva extensi&oacute;n al sistema, es importante comprobar que las bases de datos de las centralitas que forman el cl&uacute;ster estén todas
```

ellas operativas. Esto se debe a que hay que tener una réplica exacta de las bases de datos para poder registrar usuarios en cualquiera de las centralitas.

  Después de introducir una nueva extensión sería recomendable crear una nueva entrada de DialPlan para la nueva extensión creada. Si no hacemos esto, la extensión podrá registrarse pero no podrá realizar ni recibir llamadas.

  Si en algún momento alguna de las bases de datos queda fuera de servicio, y con ello su centralita Asterisk, es importante que cuando vuelva a estar operativa vayamos a la sección "Resetear BBDD" para que la tabla de usuarios de esa centralita deje de nuevo las extensiones con dirección IP a 0.0.0.0 y puerto a 0. El porqué de este paso es el siguiente: Si una base de datos queda fuera de servicio, la centralita Asterisk a la que da soporte también quedará inutilizada, por tanto, los usuarios registrados en dicha centralita buscarán soporte en otra centralita del clúster. Como la caída de la base de datos es inesperada, las direcciones IP de las extensiones registradas no se van a resetear, lo que se traduce en una mala visión de los usuarios registrados en nuestro sistema de supervisión.

4.- CrearExtensiones.jsp

Página que muestra el formulario necesario para introducir nuevas extensiones en las bases de datos:

```
<!-- PANEL CENTRAL DE RESULTADOS -->
<div class="panel">
<p> Rellene los campos que a continuaci&oacute;n se muestran para la creación de una nueva extensi&oacute;n </p>

<p> (Por lo general, el N&uacute;mero de la extensi&oacute;n y el Username deben tener el mismo valor) </p>

<form action="nuevaExtension" method ="post" >

        <p> <label for="extension">N&uacute;mero de la extensi&oacute;n :</label>
        <input type ="text" name="extension" id="extension" /> </p>

        <p> <label for="secret">Secret :</label>
        <input type ="password" name="secret" id="secret" /> </p>

        <p> <label for="username">Username :</label>
        <input type ="text" name="username" id="username" /> </p>

        <p><input class="submit" type="submit" value="Enviar" /></p>

</form>
</div>
```

5.- NuevasEntradas.jsp

La siguiente página nos aporta información acerca de la inserción de datos en las bases de datos. A esta página se le redirige al administrador cuando éste realiza una tarea de crear extensiones o entradas del dialplan, para informarle del estado de la inserción, ya haya sido un éxito o en su defecto hubiera fallado:

```
<!-- PANEL CENTRAL DE RESULTADOS -->
<div class="panel">
<% Vector estado = (Vector)request.getAttribute("estado");
Vector direcciones = (Vector)session.getAttribute("direcciones");
int numeroIP = estado.size();

for(int i=0; i<numeroIP; i++) {
        if(((String)estado.elementAt(i)).equals("true")) { %>
                <p> La inserci&oacute;n en la BBDD con direcci&oacute;n
                <%= (String)direcciones.elementAt(i) %> ha sido un &eacute;xito </p>
        <% } else { if(((String)estado.elementAt(i)).equals("fallo")) { %>
                <p> La BBDD con direcci&oacute;n <%= (String)direcciones.elementAt(i) %>
                 no est&aacute; disponible por lo que ha sido imposible la inserci&oacute;n </p>
        <% } else { if(((String)estado.elementAt(i)).equals("problema")) { %>
                <p> Fall&oacute; la inserci&oacute;n en la BBDD con direcci&oacute;n
                 <%= (String)direcciones.elementAt(i) %> </p>
        <% } } }

} %>

</div>
```

6.- VerUsuarios.jsp

Página encargada de mostrar al administrador mediante una tabla los usuarios conectados en las centralitas o en su defecto avisarle de que la consulta no se pudo realizar por diversas razones:

```
<!-- PANEL CENTRAL DE RESULTADOS -->
<div class="panel">
<% Vector estadoResultados = (Vector)request.getAttribute("estadoResultados");
Vector direcciones = (Vector)session.getAttribute("direcciones");
int aux = estadoResultados.size();

for(int i=0; i<aux; i=i+2) {
        if(((String)estadoResultados.elementAt(i)).equals("true")) { %>
                <p> Centralita Asterisk con direcci&oacute;n IP <%=
(String)direcciones.elementAt(i/2) %> dispone de los siguientes usuarios </p>
                 <table border=1>
                        <tr>
                                <th> <p> Extensi&oacute;n </p> </th>
                                <th> <p> Direcci&oacute;n IP </p> </th>
                                <th> <p> Puerto </p> </th>
                                <th> <p> Regseconds </p> </th>
                        </tr>
```

```
                                <% Vector datos = (Vector)estadoResultados.elementAt(i+1);
                                int numDatos = datos.size();
                                for(int j=0; j<numDatos; j++) {
                                    Vector datosFinales = (Vector)datos.elementAt(j); %>
                                    <tr>
                                            <% for(int z=0; z<4; z++) { %>
                        <td> <p> <%= (String)datosFinales.elementAt(z) %> </p> </td>
                                            <% } %>
                                    </tr>
                                <% } %>
                              </table>
                            <% } else { %>
                                    <p> Centralita Asterisk con direcci&oacute;n IP <%=
(String)direcciones.elementAt(i/2) %>
                                    no se puede realizar la consulta a su base de datos </p>
                             <% }
                    } %>

        </div>
```

7.- VerCDR.jsp

Página encargada de mostrar al administrador mediante una tabla el registro de llamadas de las centralitas o en su defecto si la consulta no ha podido realizarse:

```
<!-- PANEL CENTRAL DE RESULTADOS -->
<div class="panel">
<% Vector estadoResultados = (Vector)request.getAttribute("estadoResultados");
Vector direcciones = (Vector)session.getAttribute("direcciones");
int aux = estadoResultados.size();

for(int i=0; i<aux; i=i+2) {
        if(((String)estadoResultados.elementAt(i)).equals("true")) { %>
                <p> Centralita Asterisk con direcci&oacute;n IP <%=
(String)direcciones.elementAt(i/2) %> su tabla de llamadas es la siguiente: </p>
                 <div class="cdr">
                        <table border=1>
                            <tr>
                                <th> <p> Hora de llamada </p> </th>
                                <th> <p> Origen </p> </th>
                                <th> <p> Destino </p> </th>
                                <th> <p> Contexto llamada </p> </th>
                                <th> <p> Canal origen </p> </th>
                                <th> <p> Canal destino </p> </th>
                                <th> <p> &Uacute;ltima aplacaci&oacute;n </p> </th>
                                <th> <p> &Uacute;ltima sentencia </p> </th>
                                <th> <p> Duraci&oacute;n  </p> </th>
                                <th> <p> Estado </p> </th>
                            </tr>

                            <% Vector datos = (Vector)estadoResultados.elementAt(i+1);
                            int numDatos = datos.size();
```

```
                                for(int j=0; j<numDatos; j++) {
                                    Vector datosFinales = (Vector)datos.elementAt(j); %>
                                    <tr>
                                            <% for(int z=0; z<10; z++) { %>
                                <td> <p> <%= (String)datosFinales.elementAt(z) %> </p> </td>
                                            <% } %>
                                    </tr>
                                <% } %>
                              </table>
                              </div>
                            <% } else { %>
                                    <p> Centralita Asterisk con direcci&oacute;n IP <%=
(String)direcciones.elementAt(i/2) %>
                                    no se puede realizar la consulta a su base de datos </p>
                             <% }
                } %>


        </div>
```

8.- Resetear.jsp

Página encargada de mostrar el formulario para introducir la dirección IP de la base de datos de la cual se quiere resetear la tabla usuariosSIP:

```
<!-- PANEL CENTRAL DE RESULTADOS -->
<div class="panel">
<p> Introduzca la direcci&oacute;n IP de la centralita a la cual quiere resetear las extensiones registradas debido a una ca&iacute;da de su base de datos de forma inesperada </p>

<p> (Si lo desea, puede introducir m&aacute;s de una direcci&oacute;n IP, basta con separarlas por "," exactamente igual que al inicio de la aplicaci&oacute;n) </p>

<form action="resetearBBDD" method ="post" >

        <p> <label for="direccion"> Direccion/es IP :</label>
        <input type ="text" name="direccion" id="direccion" /> </p>

        <p><input class="submit" type="submit" value="Enviar" /></p>

</form>
</div>
```

9.- EstadoReset.jsp

La siguiente página nos aporta información acerca del reseteo en las bases de datos, de modo que informa del estado, ya haya sido un éxito o en su defecto hubiera fallado:

```
<!-- PANEL CENTRAL DE RESULTADOS -->
<div class="panel">
```

```
<% Vector estado = (Vector)request.getAttribute("estado");
int numeroIP = estado.size();

for(int i=0; i<numeroIP; i=i+2) {
        if(((String)estado.elementAt(i)).equals("true")) { %>
                <p> La tabla usuariosSIP de la base de datos con direcci&oacute;n IP <%=
(String)estado.elementAt(i+1) %> ha sido reseteada con &eacute;xito </p>
        <% } else { if(((String)estado.elementAt(i)).equals("fallo")) { %>

                <p> La base de datos con direcci&oacute;n IP <%=
(String)estado.elementAt(i+1) %> no est&aacute; disponible por lo que ha sido imposible
resetear su tabla usuariosSIP </p>

        <% } else { if(((String)estado.elementAt(i)).equals("problema")) { %>
                <p> Fall&oacute; la conexi&oacute;n a la hora de resetear la base de datos
                con direcci&oacute;n IP <%= (String)estado.elementAt(i+1) %> </p>
        <% } else { %>
                <p> La direcci&oacute;n IP "<%= (String)estado.elementAt(i+1) %>" no tiene
un formato correcto </p>
        <% }
        }
}
} %>

</div>
```

10.- Error.jsp

Esta página informa al administrador que antes de poder utilizar las opciones de la aplicación, estando en la portada de la misma, debe introducir una dirección IP para trabajar:

```
<?xml version='1.0' encoding='ISO-8859-1' ?>

<%@ page language="java" contentType="text/html" %>

<!DOCTYPE html
        PUBLIC "-//W3C//DTD XHTML 1.1//EN"
        "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd" >

<html xmlns='http://www.w3.org/1999/xhtml'>
  <head>
    <link href="estilo.css" rel="stylesheet" type="text/css"> </link>
    <title>Sistema de supervisi&oacute;n</title>
  </head>
  <body>
        <div class="prueba">
                <H1> <img src='Logo_asterisk.png'>  MONITOR DE CENTRALITAS
ASTERISK </H1>
                <hr class="linea">

        </div>
```

```
        <div class="prueba">

                        <H1>ERROR INTERNO A LA APLICACION</H1>
                        <div class="letra">
                        <p> Por favor, antes de hacer uso del sistema de supervisi&oacute;n
debe introducir la direcci&oacute;n IP de las centralitas
                        que desea monitorizar ya que es requisito indispensable para el
funcionamiento de &eacute;ste</p>
                    </div>
                        <p> <input type=button value=Atr&aacute;s onclick="history.go(-1)">
</p>
        </div>
  </body>
</html>
```